Type check

Assignment 2

文件入口：对整棵AST进行扫描，检查其中类型不一致的地方。

```
116  // public functions
117  void check_Prog(std::ostream& out, aA_program p)
118  {
119      for (auto ele : p->programElements)
120      {
121          if(ele->kind == A_programVarDeclStmtKind){
122              check_VarDecl(out, ele->u.varDeclStmt);
123          }else if (ele->kind == A_programStructDefKind){
124              check_StructDef(out, ele->u.structDef);
125          }
126      }
127
128      for (auto ele : p->programElements){
129          if(ele->kind == A_programFnDeclStmtKind){
130              check_FnDeclStmt(out, ele->u.fnDeclStmt);
131          }
132          else if (ele->kind == A_programFnDefKind){
133              check_FnDecl(out, ele->u.fnDef->fnDecl);
134          }
135      }
136
137      for (auto ele : p->programElements){
138          if(ele->kind == A_programFnDefKind){
139              check_FnDef(out, ele->u.fnDef);
140          }
141          else if (ele->kind == A_programNullStmtKind){
142              // do nothing
143          }
144      }
145
146      out << "Typecheck passed!" << std::endl;
147      return;
148  }
149
```

数据结构：记录扫描过的代码里，有哪些 token 及它们的类型。

```
38  struct tc_type_{
39      aA_type type;
40      uint isVarArrFunc; // 0 for scalar, 1 for array, 2 for function
41  };
```

```
// token id to token type, including function name to return type
typedef struct tc_type_* tc_type;
typedef std::unordered_map<string, tc_type> typeMap;

// func name to params
typedef std::unordered_map<string, vector<aA_varDecl>*> paramMemberMap;
```

```
//global tabels
//typeMap func2retType; // function name to return type

// global token ids to type
typeMap g_token2Type;

// local token ids to type, since func param can override global param
typeMap funcparam_token2Type;
vector<typeMap*> local_token2Type;


paramMemberMap func2Param;
paramMemberMap struct2Members;
```

print_token_map：打印 token map。也许可以帮助你进行 debug

```
26  void print_token_map(typeMap* map){
27      for(auto it = map->begin(); it != map->end(); it++){
28          std::cout << it->first << " : ";
29          switch (it->second->type->type)
30          {
31          case A_dataType::A_nativeTypeKind:
32              switch (it->second->type->u.nativeType)
33              {
34              case A_nativeType::A_intTypeKind:
35                  std::cout << "int";
36                  break;
37              default:
38                  break;
39              }
40              break;
41          case A_dataType::A_structTypeKind:
42              std::cout << *(it->second->type->u.structType);
43              break;
44          default:
45              break;
46          }
47          switch(it->second->isVarArrFunc){
48              case 0:
49                  std::cout << " scalar";
50                  break;
51              case 1:
52                  std::cout << " array";
53                  break;
54              case 2:
55                  std::cout << " function";
56                  break;
57          }
58          std::cout << std::endl;
59      }
60  }
61
62
63  void print_token_maps(){
64      std::cout << "global token2Type:" << std::endl;
65      print_token_map(&g_token2Type);
66      std::cout << "local token2Type:" << std::endl;
67      print_token_map(&funcparam_token2Type);
68  }
```

类型检查规则：在 markdown 文件里详述。

```
26  void print_token_map(typeMap* map){
27      for(auto it = map->begin(); it != map->end(); it++){
28          std::cout << it->first << " : ";
29          switch (it->second->type->type)
30          {
31          case A_dataType::A_nativeTypeKind:
32              switch (it->second->type->u.nativeType)
33              {
34              case A_nativeType::A_intTypeKind:
35                  std::cout << "int";
36                  break;
37              default:
38                  break;
39              }
40              break;
41          case A_dataType::A_structTypeKind:
42              std::cout << *(it->second->type->u.structType);
43              break;
44          default:
45              break;
46          }
47          switch(it->second->isVarArrFunc){
48              case 0:
49                  std::cout << " scalar";
50                  break;
51              case 1:
52                  std::cout << " array";
53                  break;
54              case 2:
55                  std::cout << " function";
56                  break;
57          }
58          std::cout << std::endl;
59      }
60  }
61
62
63  void print_token_maps(){
64      std::cout << "global token2Type:" << std::endl;
65      print_token_map(&g_token2Type);
66      std::cout << "local token2Type:" << std::endl;
67      print_token_map(&funcparam_token2Type);
68  }
```

一个示例实现:

```
287  void check_FnDecl(std::ostream& out, aA_fnDecl fd)
293      // if already declared, should match
294      if (func2Param.find(name) != func2Param.end()){
295          // is function ret val matches
296          if(!comp_aA_type(get_token_type(name)->type, fd->type))
297              error_print(out, fd->pos, "The function return type doesn't match the
                 declaration!");
298          // is function params matches decl
299          if(func2Param[name]->size() != fd->paramDecl->varDecls.size())
300              error_print(out, fd->pos, "The function param list doesn't match the
                 declaration!");
301          for (int i = 0; i<func2Param[name]->size(); i++){
302              if(!comp_aA_type(func2Param[name]->at(i)->u.declScalar->type,
                 fd->paramDecl->varDecls[i]->u.declScalar->type))
303                  error_print(out, fd->pos, "The function param type doesn't match the
                     declaration!");
304          }
305      }else{
```