CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
PROZ notes
Symbol table

What does an entry in your symbol table look like?
Symbol table

What information do you need to store for each of the following?

variable
array type
record type
function type
function definition
primitive type
Symbol table structure

define DeclarationType {
  PRIMITIVE_TYPE_DECLARATION,
  ARRAY_TYPE_DECLARATION,
  RECORD_TYPE_DECLARATION,
  FUNCTION_TYPE_DECLARATION,
  FUNCTION_DEFINITION,
  VARIABLE_DECLARATION
};

struct SymbolTableEntry {
  char * name;
  enum DeclarationType variant;
  union {
    struct PrimitiveType pt;
    struct ArrayType at;
    struct RecordType rt;
    struct FunctionType ft;
    struct FunctionDefinition fd;
    struct VariableDeclaration vd;
  } entry;
};
Symbol table structure

```c
enum DeclarationType { PRIMITIVE_TYPE_DECLARATION,
ARRAY_TYPE_DECLARATION, RECORD_TYPE_DECLARATION,
FUNCTION_TYPE_DECLARATION, FUNCTION_DEFINITION,
VARIABLE_DECLARATION }

struct SymbolTableEntry {
    char * name;
    enum DeclarationType  variant;
    union {
        struct PrimitiveType pt;
        struct ArrayType at;
        struct RecordType rt;
        struct FunctionType ft;
        struct FunctionDefinition fd;
        struct VariableDeclaration vd;
    } entry;
};
```
struct PrimitiveType {
  int    size; // in bytes
};

struct ArrayType {
  int    size;
  int    dimensions;
  struct VariableDeclaration declaration;
};

struct RecordType {
  int    size;
  struct SymbolTable * declaration_list // use a symbol table!
};

struct FunctionType {
  int    size;
  struct SymbolTable * domainType; // use a symbol table!
  struct SymbolTableEntry * rangeType;
};

struct FunctionDefinition {
  struct SymbolTableEntry * type; // this will be a FunctionType
  int    codePointer; // type may not be correct
};

struct VariableDeclaration {
  int    size;    // size == |primitive type|, |pointer| for array, record, function
  int    offset;  // used to determine layout of records, and stack frames
  struct SymbolTableEntry * type;
  bool   parameter; // true --> parameter, false --> local (non-parameter)
  bool   hasInitialization;
  struct ConstantValue * initializationValue;
};
```c
using union

void addEntryToSymbolTable(struct SymbolTable * st, 
                          struct SymbolTableEntry * e) {
    if (FLAG_st != 0) {
        printf("%16s : %5d : %16s : %10s \n",
               getName(e),getNumber(st),getType(e),getAnnotation(e));
    }
    st->entries = newSymbolTableEntryNode(e, st->entries);
}
```
How is `getType` defined?

```c
void addEntryToSymbolTable(struct SymbolTable * st,
                            struct SymbolTableEntry * e) {
    if (FLAG_st != 0) {
        printf("%16s : %5d : %16s : %10s \n",
                getName(e), getNumber(st), getType(e), getAnnotation(e));
    }
    st->entries = newSymbolTableEntryNode(e, st->entries);
}
```
char * getType(struct SymbolTableEntry * e) {
    if (e == NULL) {
        printf("** internal compiler error: getting type of NULL SymbolTableEntry");
        return "** NULL";
    }

    switch (e->variant) {
    case PRIMITIVE_TYPE_DECLARATION:
        return e->name;
    case ARRAY_TYPE_DECLARATION: {
        char * domainType = arrayDomain2string(e->entry.at.dimensions);
        char * map = " -> ";
        char * rangeType = e->entry.at.declaration.type->name;
        char * answer = (char *) malloc((strlen(domainType)+strlen(map)+strlen(rangeType)+1)* sizeof(*answer));
        answer[0] = '\0';
        sprintf(answer, "%s%s%s", domainType, map, rangeType);
        return answer; }
    case RECORD_TYPE_DECLARATION: {
        char * answer = cartesianProduct2string(e->entry.rt.declaration_list);
        return answer; }
    case FUNCTION_TYPE_DECLARATION: {
        char * domainType = cartesianProduct2string(e->entry.ft.domainType);
        char * map = " -> ";
        char * rangeType = e->entry.ft.rangeType->name;
        char * answer = (char *) malloc((strlen(domainType)+strlen(map)+strlen(rangeType)+1)* sizeof(*answer));
        answer[0] = '\0';
        sprintf(answer, "%s%s%s", domainType, map, rangeType);
        return answer; }
    case FUNCTION_DEFINITION:
        return e->entry.fd.type->name;
    case VARIABLE_DECLARATION:
        return e->entry.vd.type->name;
    default:
        printf("** internal compiler error: illegal variant used in SymbolTableEntry");
        return "** NULL";
    }
}
constructing SymbolTableEntries

struct SymbolTableEntry * newPrimitiveEntry(char * id, int size);

struct SymbolTableEntry * newArrayEntry(char * id, int dimensions, struct SymbolTableEntry * type, struct ConstantValue * initialization);

struct SymbolTableEntry * newRecordEntry(char * id);

struct SymbolTableEntry * newFunctionTypeEntry(char * id, struct SymbolTable * domainType, struct SymbolTableEntry * rangeType);

struct SymbolTableEntry * newFunctionDefinitionEntry(char * id, struct SymbolTableEntry * type, int codePointer);

struct SymbolTableEntry * newLocalEntry(char * id, struct SymbolTableEntry * type, struct ConstantValue * initialization);

struct SymbolTableEntry * newParameterEntry(char * id, struct SymbolTableEntry * type);
use in grammar rules

parameter_declaration:
    identifier
    COLON
    ID
{
    struct SymbolTableEntry * type = findInTable(symbolTable,$1);
    struct SymbolTableEntry * entry = newParameterEntry($3,type);
    addEntryToSymbolTable(symbolTable,entry);
}
;
use in grammar rules

identifier_list:
   ID
   initialization
   COMMA
   {
      insertLocalVariableInSymbolTable($<t>0, $2, $1);
      $<t>$ = $<t>0;
   }
   identifier_list
   |
   ID
   initialization
   {
      insertLocalVariableInSymbolInSymbolTable($<t>0, $2, $1);
   }
;
```c
void insertLocalVariableInSymbolTable(struct SymbolTableEntry * type,
                    struct ConstantValue * initialization,
                    char * id) {
    if (type != NULL) {
        addEntryToSymbolTable(symbolTable, newLocalEntry(id, type, initialization));
    } else {
        char * str1 = "the name '";
        char * str2 = id;
        char * str3 = " is being declared with an unknown type.";
        char * msg = (char *) malloc((strlen(str1)+strlen(str2)+strlen(str3)+1) *
                                      sizeof(*msg));
        msg[0] = '\0';
        sprintf(msg, "%s%s%s", str1, str2, str3);
        yyerror(msg);
    }
}
```