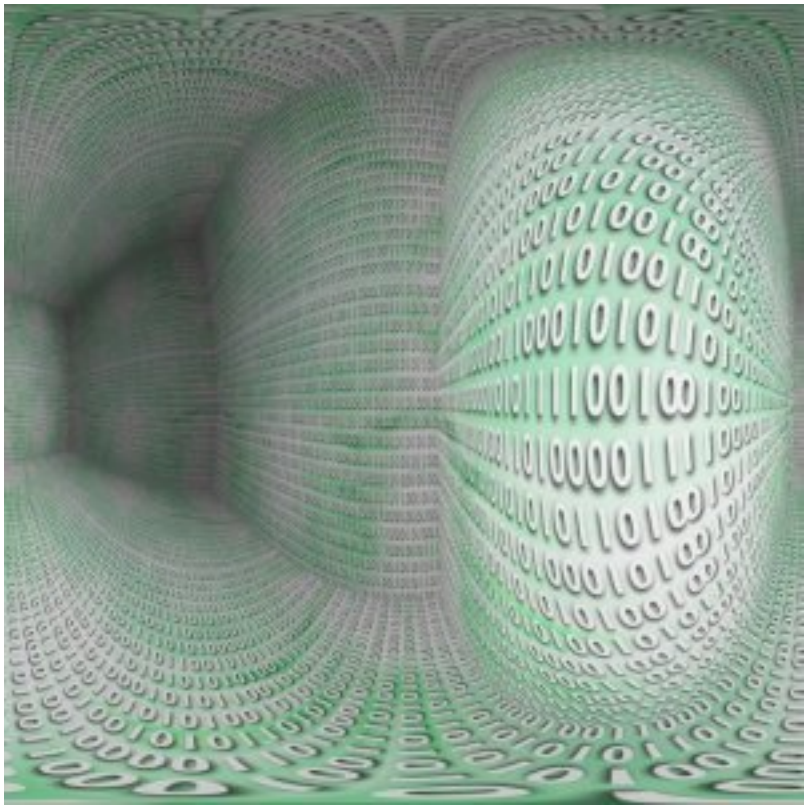
Binary Data Processing Language

Project Report



Aditi Rajoriya (ar2630@columbia.edu)

Akshay Pundle (ap2503@columbia.edu)

Bharadwaj Vellore (vrb2102@columbia.edu)

Preethi Narayan (pn2156@columbia.edu)

Table of Contents

1 Introduction

1.1 Motivation

BDPL is a programming language for processing and manipulating binary files. Specifically, the language is designed to enable a programmer to model binary file formats, read, process and write data with efficacy. In other words, the language empowers a programmer to specify a file format as one or more data structures and use these user-defined data types to parse the contents of a file.

Programming languages normally deal with files in only the ASCII or UNICODE formats. Each file is treated as a sequence of characters alpha-numeric and non-printable characters. However, binary files are not normally processed with ease by such languages. Existing languages that provide support to use system functions to read and write binary files typically require the programmer to process raw data from memory buffers into which data is read from the file.

Increasingly, with the proliferation of multimedia in multiple formats, the need is felt for tools that will enable format interchange and multi-format data packaging for distribution through various media. This provides an excellent case for a language that allows the representation of binary file formats in a manner that allows ease of reading and writing files without the programmer having to get into the nitty-gritties of bit-level operations. The objective of BDPL is exactly this – to provide a high-level language scheme to describe the layout of, access and modify binary data elements.

Bdpl was motivated by the different file formats which are available and the structures of each one them. The file formats of Mp3, Tiff, elf gave us the reason to investigate and create a language which dealt with manipulation of binary data.

1.2 Description

Bdpl is designed to be a language for simple processing and manipulation of binary data. It essentially allows any programmer with the knowledge of file formats to extract and process any data relevant to his purpose. It provides powerful constructs to read from and process files.

Bdpl is used mainly for reading files and analyzing various aspects of the files. It also enables more advanced programmers to create programs to simulate the behavior of existing programs like objdump and customise it to the requirements of the user.

Bdpl is a generic language. It provides constructs which are more powerful than the existing languages. It is not tied to any particular file format or family of formats. It is consistent in structure. Its types and operations are orthogonal. It avoids special cases.

1.2.1 Language Features

Primitive Data Types:

There are 3 basic types in Bdpl. They are:

- bit is a single bit of data
- byte is a sequence of 8 bits
- int is a 32 bit integer represented in a 2's complement form

Key words

All the key words are written in the lower case. For example, “file”, “struct” and so on. The complete listing of the keywords is given in the language reference manual.

Comparative and Mathematical Operators

Operators are necessary for condition checking and calculations. Bdpl has all the essential comparative and mathematical operators, for examples, = =, ! =, >=, =<, +, -, *, /, <, >, = and so on.

Control Statements

To control program’s flow and carry out certain operations, we will implement and support the following control statements:

• for loop: for example, we can use a for loop to read and display the values in a structure

∀if/else condition: for example, can be used to check the valid condition

1.2.2 Special features of Bdpl

Derived Data types

One of the main features of Bdpl is the the derived data types which are structures and the arrays. Structures and arrays can be used to represent entire files and they play a significant role in Bdpl. Structures are used defined data types which in turn can contain any of the basic types or structures themselves. Arrays are another set of user defined data types which contain a number of similarly “typed” data.

So we can have arrays of structures also. Bdpl restricts array representaion to one dimensional arrays. So the structures can have arrays of basic types or arrays of structures as part of their definition. Some example declarations are shown below.

```
struct outer
{
```

```
        byte size;
        struct inner
        {
              byte[10+size/2] n_bytes;
        }[*] c;
        type struct inner inner_var;

}[*] b;
```

Here b is an array of structures of the type outer. The structure itself has a member called size which is of the type byte and another structure called inner. An array of the of the inner structure is a member of each of the data type struct outer. Along with this the third member is another data node of the type struct inner.

Constraints

For every variable that is declared in a BDPL program, constraints can be specified using the satisfies-then-else clause. This satisfies clause is any expression that can be an expression in any BDPL variable of that program which is visible as per scope rules. The expression is evaluated every time that the variable is used in an assignment statement as an l-value. The BDPL program may contain statements that are conditionally evaluated when the satisfies clause is true and when it is false.

Unlimited sized arrays

One of the interesting things about arrays in Bdpl is that arrays of unlimited sizes can be created. This is done by using a "*" as the size. As example of such a declaration is as shown in the previous example for the declaration of a struct. This essentially means that arrays of infinite sizes can be made. But the philosophy behind using this concept was that data in the binary form can be voluminous and not all time do we have the specification of the size which declaration or evaluation of arrays. This allows us a mechanism deal with data when the size in not known.

When the data nodes are being populated with the data in case of unlimited arrays each time a value is to be filled a new data node of the particular type is created and the size is kept track of. So the size of the array grows automatically.

Automatic iteration in the data nodes

All processing or evaluation in Bdpl is done on the data nodes into which data has been read. So essentially a mechanism to automatically iterate over the data nodes so as to be able to perform the operations on them is required. Bdpl does this internally and processes the children nodes under each derived data type. For example to read a file into a stucture defined the data read has to be into each of the members of the structure. This is done automatically as a feature of the data node itself and the data is populated. The algorithm for that can be considered as shown below.

```
while(data)
{
put data in structure
}
```

Fieldsize

Although unlimited sized arrays can be used to read data, there are situations where although the size of the array is not known a limit to the data read can be placed. This is achieved using feildsize which places a constraint as to the maximum size a particular data type can have. This in conjunction with the unlimited sized arrays can be used to read data in which ever manner required and provides flexibility to the process of reading data.

Delayed Evaluation

Some of the main challenges we have had to deal with are complicated declaration syntax because of the use of a number of optional fields. Because of the existence of the optional fields such as fieldsize in the declaration of variables we use a mechanism of “delayed evaluation” . This involved checking whether the value of a particular data type exceeded the specified limitation.

The problem faced here was the actual evaluation using the declaration. Also fieldsize is not necessarily a constant. In many file formats there are situations where the size of headers etc are calculated based on the other values read. So this meant that fieldsize can also be an expression. So be able to implement this a context of the data node being created had to be stored in the symbol table along with a pointer to the AST where the fieldsize was indicated. So while executing portions of the program where this data type was used the context could be obtained and evaluated to see whether there is a conformance of the usage of the data node. This is a mechanism of “delayed evaluation” used in Bdpl.

So Bdpl provides a mechanism of evaluating code during declaration by the allowance of declaration in conjunction with fieldsize.

1.2.3 Design Goals

In Bdpl all data types are constructed in a manner such that they are treated as first order classes . Manipulation of all the data in either structures or arrays can be made similar to that of basic types. It is a robust language and allows a wide variety of operations to be performed so as handle binary data.

1.2.4 Sample Programs

Given below is a simple program in Bdpl to calculate the GCD of 2 given numbers.

```
int a; int b;
a = 96; b = 36;
for (;;) {
      if(a > b)   {
            a = a - b;
      }
      else if(a < b)    {
            b = b - a;
      }else {
            break;
      }
}
print(a);//Find the GCD of two integers
```

2 Language Tutorial

Running the compiler

```
java BdplMain -f <filename>
```

Bdpl programs consist of a sequence of decelerations followed by a sequence of statements. A very simple Bdpl program is :

```
print("hello world");
```

The above is a complete Bdpl program. It has only one statement, which is an executable statement asking Bdpl to print the string “hello world” to standard output. In Bdpl , you can print constant strings, data of basic types and data of structures / arrays. So if you wanted to print the sum of two numbers, you could use the following program:

```
int a;
int b;
a=21;
b=21;
print(a+b);
```

Which should print out 42 as the answer.

Bdpl has a for statement to support iteration. The following program prints the first 25 Fibonacci numbers :

```
int a; int b;
int c; int i;
a = 1; b = 1;
for(i=0;i<25;i=i+1)
{
      print(a); print("::");
      c = a + b;
      a = b;
      b = c;
}
```

Bdpl allows you to define and print our structures. Here is a program that defines a structure of 2 integers and prints it out . The output is given next to the program.

```
struct A
{
      int a1;
      int a2;
}a;

a.a1=1;
a.a2=2;

print(a);
```

```
Output:

struct:A
{
        a1 => 1,
        a2 => 2,
}
```

You can read structures directly from files in Bdpl. The following program demonstrates how to read the structure defined above from a file. The program opens the file name test.dat, gets 8 bytes of data and makes the value of a1 equal to the first 4 bytes and a2 , the next 4.

```
struct A
{
      int a1;
      int a2;
}a;
file "x:/test/test.dat" inp_file;
read(inp_file,a);
print(a);
```

```
Contents of test.dat (in hex) :
0000 000d 0000 000d

Output :
struct:A
{
        a1 => 13,
        a2 => 13,
}
```

Bdpl allows you to define arrays of structures and then read these arrays from a file too. Thus, if you had a file in a particular format, you can model the format as a Bdpl structure and read from the file. But what if you didnt know the number of elements in the file ? In Bdpl, you can indicate that the length of an array is not known by marking it's length as “*” . The length of the array will then depend on how much input is available. The following programs reads all pairs of bytes from a file. If the filesize is odd number of bytes, the last byte wont be read.

```
struct A
{
      byte a1;
      byte a2;
}[*] a;

file "x:/test/test.dat" inp_file;
read(inp_file,a);
print(a);
```

```
ARRAY of struct:A
[
        0 => struct:A
        {
                a1 => 0x61,
                a2 => 0x62,
        }
        ,
        1 => struct:A
        {
```

```
Contents of test.dat (in hex) :
6162 6364 0d0a
```

```
                                        a1 => 0x63,
                                        a2 => 0x64,
                                }
                                ,
                                2 => struct:A
                                {
                                        a1 => 0xd,
                                        a2 => 0xa,
                                }
                                ,
                        ]
```

You can also test whether each structure is a valis structure or not. Eg, suppose you always wanted a1 to be a valid character in the range of 'a' to 'z' (ascii 97 to 122), we can say that a1 “satisfies” the given condition. In Bdpl, while reading structures, you can automatically test whether the data “satisfies” a condition and execute code if it doesn't. We modify the above program to see if a1 is always a lowercase alphabet character (ascii 97 to 122). We print “valid” if the condition is satisfied and “invalid” otherwise. The inputs to the program are the same, so the first structure is valid and the next 2 are invalid as can be seen from the output.

```
struct A
{
      byte a1 satisfies {a1>=97 && a1<=122}
            then
            {
                  print("Structure ");
                  print($#a);
                  print(" valid ::" );
            }
            else
            {
                  print("Structure ");
                  print($#a);
                  print(" invalid ::" );
            };
      byte a2;
}[*] a;

file "x:/test/test.dat" inp_file;
read(inp_file,a);
```

```
Output:
Strucutre 1 valid
Strucutre 2 valid
Strucutre 3 invalid
```

Notice that a1 magically became the a1 that was being read at that time. Notice also the weird $#a in the print statement. This inbuilt variable refers to the number of elements in an array. Thus, while the array a is being read, the size of the array will grow dynamically. $#a changes to indicate this.

3 Language Reference Manual

3.1 Lexical conventions

BDPL comprises the following types of tokens – keywords, identifiers, constants, operators and separators. The language is a free-form language; spaces, tabs and newlines only serve to separate tokens of the language. If the input stream has been parsed into tokens up to a certain character, the next token is taken to include the longest string of characters that could possibly constitute a token.

3.1.1 Comments

Comments are always single-line only. The characters // begin a comment which terminates at the end of the line.

3.1.2 Identifiers

An identifier is a sequence of letters, digits and the underscore character, with the first character being either a letter or an underscore('_'). Letters are case sensitive; an upper case character counts as different from a lower case character. Only the first eight characters of identifiers are significant.

3.1.3 Keywords

The following identifiers are reserved for use as keywords and may not be used otherwise:

```
 bit          byte       else        exit        fieldsize   for          if
printstring   print      read        file        then        else       satisfies
 struct       this       type
```

3.1.4 Reserved Words

The following strings are not identifiers in BDPL. However, they are reserved for future use as keywords and therefore, should not be used as identifiers in the language.

```
 uimsbf      imsbf       uilsbf      Ilsbf      on      optional
```

3.1.5 Constants

The different types of constants in BDPL are :

Decimal constants

A decimal constant is any sequence of digits.

Hexadecimal constants

A hexadecimal constant begins with the character sequences '0' 'x' or '0' 'X', and comprises any sequence of digits, lower case letters 'a' to 'f' and upper case letters 'A' to 'F' thereafter.

Binary constants

A binary constant begins with the character sequences '0' 'b' or '0' 'B', and comprises any sequence of the binary digits '1' and '0' thereafter.

3.2 Types

Every BDPL identifier has a 'type' which determines the meaning of the value stored in the identifier's storage.

3.2.1 Basic Types

There are these basic types in BDPL:

`bit` is a single bit of data

`byte` is a sequence of 8 bits

int is a 32 bit integer represented in a 2's complement form

More types can be derived from these basic types using the following constructs:

`arrays`, which are sequences of elements of the same type

`structures`, which contain elements of various types

3.2.2 Arrays

An array is a sequence of elements of the same type. Arrays can be formed from basic types or from structures (defined later). Individual elements of an array can be addressed by specifying the “index” of the element. An index of an element is the position of the element from the start.

Suppose A is an array, then A[i] addresses the element at a distance of i elements from the start of the array. Thus, A[0] denotes the first element of the array (or equivalently, an element at a distance 0 from the start of the array). The type of the elements of A determines whether all elements are of the same size. This may not be the case if the array is of structures. Structures of the same type can have variable sizes (as we will see in the next section), and thus, each array element may be of a different size.

The concept of arrays in BDPL is illustrated by the following example of an array declaration:

```
byte[10] ar1,ar2;
```

This declaration defines two arrays, a1 and a2, each of which comprises 10 bytes. The expression ar1[3]

refers to the fourth element of ar1 (or equivalently, an element at a distance of 3 elements form the start of ar1).

The size of an array need not be specified at the time of declaration. BDPL provides a special notation to indicate that the number of elements in the array is not known as compile time and that it will be determined at run time. The declaration:

```
byte[*] ar1;
```

defines the array `ar1` whose size will be determined at run time.

When an array appears without an index (or a set of indices), it is assumed that the complete bit-sequence of the array is addressed. For instance, `A = 0` sets the numerical value represented by the entire array of elements taken together to 0. Only one dimensional arrays are permitted.

3.2.3 Structures

Structures (or structs) in BDPL are defined using the “struct “ keyword. Structs are the basic facility that BDPL provides in modeling the structure of a file. A struct represents an abstract entity that defines the layout of a new type which comprises basic types, arrays of defined types(basic types as well as structs) and structs, and models the fields in the file being processed. When the contents of a file are read into a struct of a defined type, the number of bytes corresponding to the size of the struct is read from the file and interpreted as per the types of the contained fields or members. Once a struct is defined, it is reckoned a new type and can be used in operations exactly like a basic type.

The definition of a struct is done thus:

```
struct a
{   .... }x;
```

This defines a struct named “a”, which means “a” is a new type which can be used in the remainder of the BDPL program. The body of the struct, comprising valid declarations, appears between the curly braces.

The struct represents a sequence of elements of possibly heterogeneous types. The order of declarations that occur in the structure is very important. Therefore,

```
struct A
{
      int a;
      byte[10] b;
}_A;
```

is different from

```
struct A
{
      byte[10] b;
      int a;
}_A;
```

Every element of the struct is addressable by name. Elements of a structure are referenced using the '.'

operator. For instance, the expression `_A.b` references the element named b in the structure named _A. The left operand of the '.' operator identifies the container, and the right operand identifies the contained. The name of the struct itself is used in conjunction with “type struct” to denote the type defined by the structure.

The bit sequence associated with the struct in memory is the concatenation of the bit-sequences of its individual elements.

3.3 Files

Files are special types in BDPL, and a file instance is created by a declaration comprising the keyword `file`. For each file that is read from or written into in a program, there may be an independent file instance. Once a file instance is created, it must be associated with content from a real file. This is done via a read statement. Also, once data corresponding to a file instance is ready, it can be committed to a real file. This is done via the write statement. Statements that operate on files are detailed later. This section deals with files and their properties.

An element that is declared to be of a file type may be perceived as a file buffer. Conceptually, this is equivalent to the following structure.

```
struct file {
      byte[*] data;
}_file;
```

File variables are declared as follows:

```
file-declaration:
      file filename-list
filename-list:
      filename
      filename-list
filename:
      file_path identifier
```

Files, once declared, may be used in assignment statements, read and set statements. A file is considered to be a raw sequence of bits. Hence when assigning a file to a structure, the types of the structure elements dictate the interpretation of the raw data in the conceptual file buffer. Also, when assigning structures to files, the types of the structure's elements dictates the format of the bytes that are populated in the conceptual file buffer.

3.4 Lvalues

An lvalue is an expression which refers to an object to which a value can be assigned. In other words, and lvalue is an expression that can occur on the left hand side of a BDPL assignment statement. Any identifier of

a basic type, an array name, one or more indexed elements of an array, a struct type element, or a member of a struct may form an lvalue. The nature of type conversion that occurs for various lvalue expressions is detailed in the sections on assignment statements and type conversions.

3.5 Conversions

The operators in BDPL may, depending on the types of the operands, convert an operand from one type to another. All types conversions are implicit. The nature of type conversion or type promotion that is adopted for all operands is detailed in the table below and governed by the following rules.

1. All types other than bit[], byte[] and struct are regarded as types that possibly carry numeric values. Hence in all conversions involving these types, a best attempt is made to maintain the numeric value following conversion.
2. Bit-Order: In an array of bits, the left-most bit (index 0) is regarded as the most significant when all bits are taken together.
3. Endian-ness: In an array of bytes, the byte at index 0 is regarded as least significant when all bytes are taken together.
4. In an assignment expression, the type of the lvalue is held sacrosanct and the type of the right hand side is converted to the type of the left-hand side as per the type conversion table.
5. When converting a bit-sequence to an array of undefined size, with the with the whole array taken at once,
    1. the size of the array is taken to be the fieldsize if specified
    2. the size of the array is taken to be the size of the right-hand size expression rounded upwards to be a multiple of the size of the basic type from which the array is formed.
6. Expanding: When converting a bit-sequence to one of a larger size, the bit sequence is copied at the least-significant end of the target sequence, and sign-extension rules are applied if required.
7. Truncating: When converting a bit-sequence to one of a smaller size, the bit sequence is truncated to the required size by retaining the least significant end and chopping of the most significant bits that cannot be accommodated.
8. Sign extension is performed only when both the type from which conversion is being done and the type to which conversion is being done are signed.
9. When dealing with expressions where the types on the LHS and the RHS are * arrays (something like bit[*] = bit[*]) , everything form the RHS is populated into the first element of the LHS.

		FROM						
		bit	byte	int	bit[]	byte[]	struct	
TO	bit	Copy bit	Copy LSb	Copy LSb	Copy LSb	Copy LSb of byte[0]	Copy next bit	
	byte	Copy bit to LSb, NSE	Copy byte	Copy LSB	Copy 8 LSb, T, *, NSE	Copy LSB, *	Copy next byte	
	int	Copy bit to LSb, NSE	Copy byte, SE	Copy bytes	Copy 32 LSb, T, *, E, NSE	Copy 4 LSB, T, *, E, NSE	Copy next 4 bytes, T, E	
	bit[]	Copy bit to LSb, *	Copy to LSB, T, *	Copy to 32 LSb, T, *, E	Copy LSb to LSB, T, *, NSE	Copy LSb to LSb, T, *, NSE	Copy next bits, *	
	byte[]	Copy bit to LSb of byte[0]	Copy to LSB, *	Copy to 4 LSB, T, E, *	Copy LSb to LSB, T, *, NSE	Copy LSB to LSB, T, *	Copy next bytes, *	
	struct	Error	Error	Error	Copy bits, *	Copy bits, *	Type cast	

Notation	Meaning
LSb	Least Significant Bit
LSB	Least Significant Byte
NSE	No Sign Extension
SE	Sign Extension as per rules for extension
*	Follow rules for handling arrays of undefined size
T	Follow rules for truncation and expansion
E	Follow rules for endian-ness

When the operands in an expression are of different types, the following rules are applied to promote one type to the other prior to performing the operation. Note that these rules do not apply to the assignment and push operators. In the case of the assignment and push operators, the type of the lvalue expression is held sacrosanct, and type conversion is applied to the expression on the right-hand side. To convert, the rules defined in the table above are used.

1. If one of the operands is a bit and the other is either a byte, int, float or double, the bit is promoted to the other type for purposes of the operation.

2. If one of the operands is a byte and the other is either an int, float or double the byte is promoted to the other type for purposes of the operation.

3. If one of the operands is an int and the other is either a float or a double, the int is promoted to the other type for purposes of the operation.

4. If one of the operands is a float and the other, a double, the float is promoted to a double.

Bit array and byte array operands may only be used in assignment statements and shift/rotate operations. Type promotion is not relevant for the latter. In the former case, standard type conversion rules are applied.

3.6 Expressions

Expressions are ordered by precedence starting from the highest precedence.

3.6.1 Primary Expressions

Identifier

primary-expr: identifier

An identifier is a primary expression. Each identifier must be declared to be of a basic or defined type.

Numeric Constant

A numeric constant may be an decimal, binary or hexadecimal number.

Parenthesized Expressions

expr: (expr)

A parenthesized expression is a primary expression whose type and value are identical to those of the unadorned expression. The presence of parentheses does not affect whether the expression is an lvalue.

Array expression

```
expr:       primary-expr[expr]
            primary-expr[expr-list]
expr-list:  expr
            expr,expr-list
```

An array expression is also an expression. The methods of indexing the elements of the array is already detailed in section 3.2.2.

Dot Operator

primary-expr: primary-expr.member

A primary expression followed by a dot followed by the name of the member of a structure is a primary expression. The element `member` must be defined to be a field within the structure which is being referenced.

3.6.2 Unary Operators

Associativity : All operators described in this section are right associative

Negation operators

```
expr :      ~ expr
            ! expr
```

There are two negation operators, Bitwise negation (~) and Logical negation (!). These are both unary operators. The logical negation operator logically negates expressions. Ie, converts a non-zero value to zero and a zero value to 1 . The bitwise negation operator does a bitwise flipping of bits so that ~0b1101 becomes 0b0010.

Offset Operator

There are 3 offset operators, Position operator ($) , Size operator ($#) and Byte-size operator (#).

Size Operator

The size operator ($#) returns the number of elements contained in the operand. This can operate only on arrays. This cannot be used as an lvalue.

3.6.3 Multiplicative Operators

```
expr : expr * expr
```

Associativity: Left to right

The '*' operator is a binary operator that indicates multiplication. The result of a multiplication expression is the product of the operands. Operands may only be of basic types. Type promotion rules are applied as detailed in the section on type conversion when the two operands are of different types.

```
expr : expr / expr
```

Associativity: Left to right

The '/' operator is a binary operator that indicates division. The result of a division expression is the quotient. Operands may only be of basic types. Type promotion rules are applied as detailed in the section on type conversion when the two operands are of different types.

```
expr : expr % expr
```

Associativity: Left to right

The '%' operator is a binary operator that indicates modulus. The result of a modulus expression is the

remainder from dividing the left operand by the right operand. Operands may only be of basic integer types. Type promotion rules are applied as detailed in the section on type conversion when the two operands are of different types.

3.6.4 Additive Operators

expr : expr + expr

Associativity: Left to right

The '+' operator is a binary operator that indicates addition. The result of an addition expression is the sum of the operands. Operands may only be of basic types. Type promotion rules are applied as detailed in the section on type conversion when the two operands are of different types.

expr : expr - expr

Associativity: Left to right

The '-' operator is a binary operator that indicates subtraction. The result of a subtraction expression is the difference of the operands. Operands may only be of basic types. Type promotion rules are applied as detailed in the section on type conversion when the two operands are of different types.

3.6.5 Shift Operators

Bitwise shift Operators

expr : expr << expr

expr : expr >> expr

Associativity: Left to right

The '<<' and '>>' operators are logical shift operators. The left operand may only be of a basic integer type or an array of these types. The second operand must be a positive integer type. The result of the expression with '<<' is the bit-sequence of the first operand is rotated left as many times as indicated by the value of the right operand. Vacated bit positions are filled with 0. The result of the expression with '>>' is the bit-sequence of the first operand is rotated right as many times as indicated by the value of the right operand. Vacated bit positions are filled with 0.

Arithmetic Shift Operators

expr : expr <<< expr

expr : expr >>> expr

Associativity: Left to right

The '<<<' and '>>>' operators are arithmetic shift operators. The left operand may only be of a basic integer type or an array of these types. The second operand must be a positive integer type. The result of the

expression with '<<<' is the bit-sequence of the first operand is rotated left as many times as indicated by the value of the right operand. Vacated bit positions are filled with 0. The result of the expression with '>>>' is the bit-sequence of the first operand is rotated right as many times as indicated by the value of the right operand. Vacated bit positions are filled with the most significant bit taken as sign. Essentially, the left operand is treated as a number in 2's complement form.

Rotation Operators

```
expr : expr <-< expr
expr : expr >-> expr
```

Associativity: Left to right

The '<-<' and '>->' operators are left and right rotation operators. The left operand may only be of a basic integer type or an array of these types. The second operand must be a positive integer type. The result of the expression with '<-<' is the bit-sequence of the first operand is rotated left as many times as indicated by the value of the right operand. Vacated bit positions are filled with the bits that are rotated out of the left of the bit-sequence. The result of the expression with '>->' is the bit-sequence of the first operand is rotated right as many times as indicated by the value of the right operand. Vacated bit positions are filled with the bits that are rotated out of the right of the bit-sequence. This operator is not yet implemented.

3.6.6 Comparison Operators

Associativity is not defined for comparison operators since expressions with these operators cannot be cascaded.

Greater than

```
expr : expr > expr
```

Less than

```
expr : expr < expr
```

Greater than or equal to

```
expr : expr >= expr
```

Less than or equal to

```
expr : expr <= expr
```

The result of these expressions is a 1 if the boolean result of the expression is true and 0 otherwise. The operands must be of a basic type.

3.6.7 Equality Operators

Equal to

expr : expr == expr

Not Equal to

expr : expr != expr

The result of these expressions is a 1 if the boolean result of the expression is true and 0 otherwise. The operands must be of a basic type.

3.6.8 Bitwise AND Operator

expr : expr & expr

Associativity: Left to right

The `&` operator performs a bitwise logical 'and' or of the operands. The operands are not required to be of the same type. However, both operands are required to be of a basic type and must have equal lengths in bits.

3.6.9 Bitwise XOR Operator

expr : expr ^ expr

Associativity: Left to right

The `^` operator performs a bitwise 'exclusive or' of the operands. The operands are not required to be of the same type. However, both operands are required to be of a basic type and must have equal lengths in bits.

3.6.10 Bitwise OR Operator

expr : expr | expr

Associativity: Left to right

The `|` operator performs a bitwise 'inclusive or' of the operands. The operands are not required to be of the same type. However, both operands are required to be of a basic type and must have equal lengths in bits.

3.6.11 Logical AND Operator

expr : expr && expr

Associativity: Left to right

The `&&` operator returns 1 if both of its operands have a non-zero value, and 0 otherwise. Each of the

operands must be of a basic type. However, both operands are not required to be of the same type. The second operand is not evaluated is the value of the first operand is zero.

3.6.12 Logical OR Operator

```
expr : expr || expr
```

Associativity: Left to right

The || operator returns 1 if either of its operands has a non-zero value, and 0 otherwise. Each of the two operands must be of a basic type. However, both operands are not required to be of the same type The second operand is not evaluated if the value of the first operand is non-zero.

3.6.13 Assignment Operator

```
object = expr
```

The assignment operator denotes that the variable on the LHS will hold the value of the expression on the RHS. Various type conversion rules will be applied to evaluate what exactly should happen when an assignment is executed. Indeed, there are many consideration of endian-ness, loss of precession, unmatched types etc. which have been address in section 5.

3.6.14 Push Operator

```
object <- expr
```

The push operator takes the expression on the right and streams the value to the lvalue. It is like an assignment, but non- destructive. The current value of the object is not overwritten. Instead, the data is made available to the object for appending. This operator is not yet implemented.

3.7 Declarations

Declarations are used to specify types for identifiers. Whether declarations reserve memory for identifiers is dependent upon the implementation of the compiler. However, once declared, an identifier refers to an object that is of the indicated type and which is available till the end of the program's execution. Declarations take the following form:

```
declaration:
    type-specifier array declarator-list valid-check optional-check
```

Each declaration statement may comprise at most one type specifier. The type specifier field may be followed by the array operator, which makes every declared element an array of the specified type. The specification of the type is followed by a list of identifiers which are named elements of the said type. This list may be followed by two optional clauses referred to in the grammar above as valid-check and optional-check. These are detailed in later sections. Each declaration statement must be terminated with a semi-

colon. A declaration statement may not appear within a statement-block.

```
declarator:
      identifier initializer fieldsize
      (declarator)
```

3.7.1 Type Specifiers

The type specifier for a declaration may be any of the basic types available in BDPL, a structure definition, or a reference to a structure that is completely defined earlier in the BDPL source file.

```
type-specifiers:
      int
      bit
      byte
      struct-defn
      struct-type
struct-defn:
      struct name {declarations}
name:
      identifier
struct-type:
      type struct name
declarations:
      declaration
      declaration declarations
```

Each defined structure must be assigned a name, which identifies the type just created by virtue of the definition. The struct-defn must be accompanied by the declaration of an element of that struct type. The struct definition may contain any number of valid declaration statements within it, including the definition of other struct types, as detailed in 3.2.3. The struct-type construct must be employed when creating elements of a struct which has previously been defined. The name must be of a struct defined earlier.

3.7.2 Array Types

Array types may be created by following the type specification for an identifier by the array operator along with a specification of the size of the array. The size of the array may be specified in the form of any valid expression. The size of the array may also be specified as unknown, in which case, a '*' character should be used in the size field. This creates a free-size array.

```
array:
      [expr]
      [*]
```

3.7.3 Identifier Properties

The each identifier may be qualified by an additional specification of its environment and properties. This is in the form of an optional initializer and an optional field-size. Currently no strings are defined.

```
initializer:
      ("string" => "string")
      ("string" => "string"),initializer
```

The field-size construct may to be used to specify circumstances wherein the application would like that information be stored in a data structure of size other than in the file being processed. For an array of unspecified size, the field-size indicates the maximum size in bits of the array element. For elements where size is not variable, the field-size indicates the size of the representation of that field in a file should the field be written to or read from a file.

```
field-size:
      fieldsize expr
```

3.7.4 Constraint Checks

For each identifier which is declared, constraints may be imposed on the value, set of values or range of values that the variable may take. The variable is checked for a valid value assignments whenever an expression with the identifier as lvalue is evaluated. The valid value check takes the following form:

```
valid-check:
      satisfies {expression} then-block_opt else-block_opt
then-block:
      then {statement-block-1}
else-block
      else {statement-block-2}
```

The value to be assigned to the identifier is evaluated and validated against the value-specifier. If this comparison indicates that the assigned value is valid, the statement block-1 is executed, if present. If the comparison fails, statement block-2 is executed if present. The ok and nok constructs are both optional and either or both of these may be skipped. The ok keyword and statement-block1 must either both be present, or neither. The nok keyword and statement-block2 must either both be present, or neither.

3.8 Statements

Statements are executed in the order in which they are written in the program, except when the order is

explicitly changed by the program itself, using the statement constructs indicated below.

3.8.1 Expressions

Construction:

```
expression;
```

An expression terminated by a semi-colon constitutes a basic statement.

3.8.2 Statement Blocks

Construction:

```
{statement-block}
      statement-block:
            statement
            statement statement-block
```

A statement block is a collection of statements that can be used together in lieu of a single statement.

3.8.3 Conditional Statements

Construction:

```
if (expression) statement else statement
if (expression) statement
```

The conditional statement can occur either with or without an else section. If the expression evaluates to a non-zero value, the first statement is executed and if the expression evaluates to zero, the second statement is executed, if present. The else is connected with the last encountered else-less if.

3.8.4 For Loop

Construction:

```
for(expression-1 ; expression-2 ; expression-3) statement
```

The loop statement is used to execute a statement several times depending upon a condition. In the said syntax, expression-1 specifies an initialization for the loop, expression-2 specifies a condition which is evaluated prior to each iteration and expression-3 specifies an incrementing action that is performed after each iteration and before the evaluation of expression-2. Any or all of the expressions may be omitted.

3.8.5 Continue

Construction:

```
continue
```

This statement must be used within the for loop. It causes control to skip over the remaining statements in the enclosing loop statement block and pass to the end of the loop.

3.8.6 Break

Construction:

```
break
```

This statement must be used within the for loop. It causes control to break out of the loop and pass to the statement immediately following the loop statement.

3.8.7 Read

Construction:

```
read "string" identifier
```

This statement allows the contents of a file to be read into a conceptual file buffer. The path to the file whose contents have to be read is supplied as a string, and forms the first element of the read statement within braces. The file type variable which represents the buffer into which the file must be read is the second element. This file element must already have been declared through a file declaration statement.

3.8.8 Print

Construction:

```
print(string)
```

This statement allows the programmer to write message to the console from within the BDPL program. The message to be written must be a string enclosed within braces that follow the print keyword.

3.8.9 Exit

Construction:

```
exit(int)
```

The execution of this statement stops execution of the current program and returns. The argument (integer) is returned by exit to the caller. This statement is not yet supported

3.9 Scope and lifetime

Bdpl has a challenging scoping scheme for variables. Every struct starts a new scope. Since we have arrays of structs, every element of an Array of structs has its own scope. Furthermore, every identifier can have code associated with it which should be executed every time the identifier is populated. For enabling this, we store a pointer to the variable symbol table for every identifier. Thus nested scoping is used to represent

nested constructs such as structures within structures and arrays of structures.

3.10 Precedence and Associativity

Operator	Name	Precedence	Associativity
[]	Aray index	1	Left
,	List separator	1	Left
$	Position	1	Right
$#	Size	1	Right
#	Byte size	1	Right
( )	Parenthesis	1	
..	Range	1	
~	Bitwise Negation	2	Right
!	Logical Negation	2	Right
*	Multiplication	3	Left
/	Division	3	Left
%	Modulo	3	Left
+	Addition	4	Left
_	Subtraction	4	Left
<<	Logical Left shift	5	Left
<<<	Arithmetic left shift	5	Left
>>	Logical Right Shift	5	Left
>>>	Arithmetic right shift	5	Left
>->	Right rotate	5	Left
<-<	Left Rotate	5	Left
<	Less than	6	Left
>	Greater than	6	Left

<=	Less than equals	6	Left
>=	Greater than equals	6	Left
==	Equals	7	Left
!=	Not equals	7	Left
&	Bitwise And	8	Left
^	Bitwise Xor	8	Left
|	Bitwise Or	8	Left
&&	Logical And	9	Left
||	Logical Or	9	Left
? :	Conditional	10	Right
=	Assignment	11	Right
<-	Push	11	Right
=>	Set	11	Right

4 Project Plan

4.1 Processes Used

4.1.1 Weekly Meetings

Once our team was formed we had regular group meetings after the classes and during the weekends to discuss the project scenarios. We divided the tasks between the team members and set the goals for every team member. Initially we had more frequent group meetings to discuss the structure and the functionality of our language. But once we were sure what our language is going to do we started working independently and updated each other using email.

4.1.2 Group Formations

To keep every member updated major communication tool used was Yahoo Mail. We formed a plt-bdpl group on the yahoo groups which contains the major updates regarding the project. Skype and Yahoo Messanger was used for online communication and conferencing. We kept a record of “todo” list and “issues” for a language as a thread on our plt-bdpl yahoo groups and Google Code, so that everyone was aware of what new issues and challenges are coming at different stages of our project and everyone can have a thought to analyze the situation.

4.1.3 Subversion

We used TortoiseSVN, a windows based subversion client that is integrated into the Windows shell.

4.1.4 Project Management Tool

We used AceProject as a Project Management Tool for our project. AceProject has plenty of project management features such as time tracking, Gantt charts, calendars and document management. It allowed us to set up our project with multiple team members.It helped in setting the deadlines, and assigning tasks to specific users. We divided our project in four parts:- Documentation, Front End, Back End and Testing Framework. We listed out the things which are of higher priority and which were of low priority. Each group member was assigned responsibility with a due date. Once the assigned tasks were finished new sets of priorities were decided and assigned to the group members. The Gantt charts for the weekly for all phases of the projects are shown below in the form of timelines. AceProject acted as the critical tool for our project organization.

4.1.5 Iterative Process

We used iterative process in our modeling. At each stage simultaneously we applied testing so that we are sure that we are moving in the right direction. We believed in testing in stages rather then building the whole

and keeping the testing for the later stages. Testing played a very important role for our project. The testing was done at all the stages of our project.

4.2 Programming Style Guide

Each individual has his/her own coding style. But we did have the following rules set for ourselves that all of us tried to follow:

- All the ANTLR grammar should be in same .g file.
- There should be clear documentation in each part of the code so that team member apart from the programmers understand it without any difficulty.
- Each new version should clearly accompany the changes/enhancements made from the previous version. This would later provide a good log for the project.
- Any major changes in design, etc ANTLR rules, should be first discussed with all the team members before making any change in the code.
- The top of every Java file should have brief comment telling about what this class does.
- Java code should follow the Java coding standards set by the Sun Microsystems.
- We made use of the empty constructors.
- The comments are given in the Javadoc format

4.3 Responsibilities Assigned

Framework	Modules	Assigned to
Front End	1.Lexer 2.Parser 3.Tree Walker	Bharadwaj Bharadwaj Bharadwaj,Akshay
Back End	1.Data Types 2.Expression Evaluation 3.Symbol Tables 4.Reading from a file 5.Type Convention 6.Type Checking 7.Scoping,Delayed Evaluation, Star Arrays	Akshay Bharadwaj Preethi,Akshay Akshay Akshay Akshay Akshay
Testing	1.Test Suite set up 2.Test cases	Bharadwaj Bharadwaj,Akshay, Preethi and Aditi
Documentation	1.LRM 2.Project Report 3.Project Presentation	Bharadwaj,Akshay, Preethi and Aditi Preethi Akshay and Bhardwaj

4.4 Project Log

Task	Due Date
Team Formation	11th Sep 2007
Brainstorming the ideas	13th Sep 2007
Exploring BDPL	18th Sep 2007
Initial Draft for Project Proposal	20th Sep 2007
Final Project Proposal Sudmitted	25th Sep 2007
Discussing Finer parts of BDPL	29th sep 2007
Exploring Nitti-Gitti's of Language	2nd Oct 2007
Started with Lexer and Parser	6th Oct 2007
Finished Lexer	12th Oct 2007
Initial Draft LRM	16th Oct 2007
Final LRM Submitted	18th Oct 2007
Midterm	25th Oct 2007
Finished Parser	1st Nov 2007
Walker Design and BDPL Tree Walker	19th Nov 2007
Finished BDPL Tree Walker	26th Nov 2007
Testing Language and Test Cases	3rd Dec 2007
Final Exams	6th Dec 2007
Investigating File Formats	10th Dec 2007
Integration and Testing	12th Dec 2007
Working on Final Report	14th Dec 2007
Working on Final Presentation	16th Dec 2007
Final Report due with Presentation	18th Dec 2007

4.5 Development Environment and Tools Used

4.5.1 Java

Programming Language.

4.5.2 Antlr

Front End/Grammer Specifications.

4.5.3 Net Beans

Java Development Environment.

4.5.4 Ant

Build Environment

4.5.5 Google Code/Tortoise

SVN based Configuration Management

4.5.6 Google Code

Issue Submissions and Tracking

4.5.7 TCL

Test Scripting/Automation

4.5.8 AceProject

Project Planning and Tracking

4.5.9 YDoc

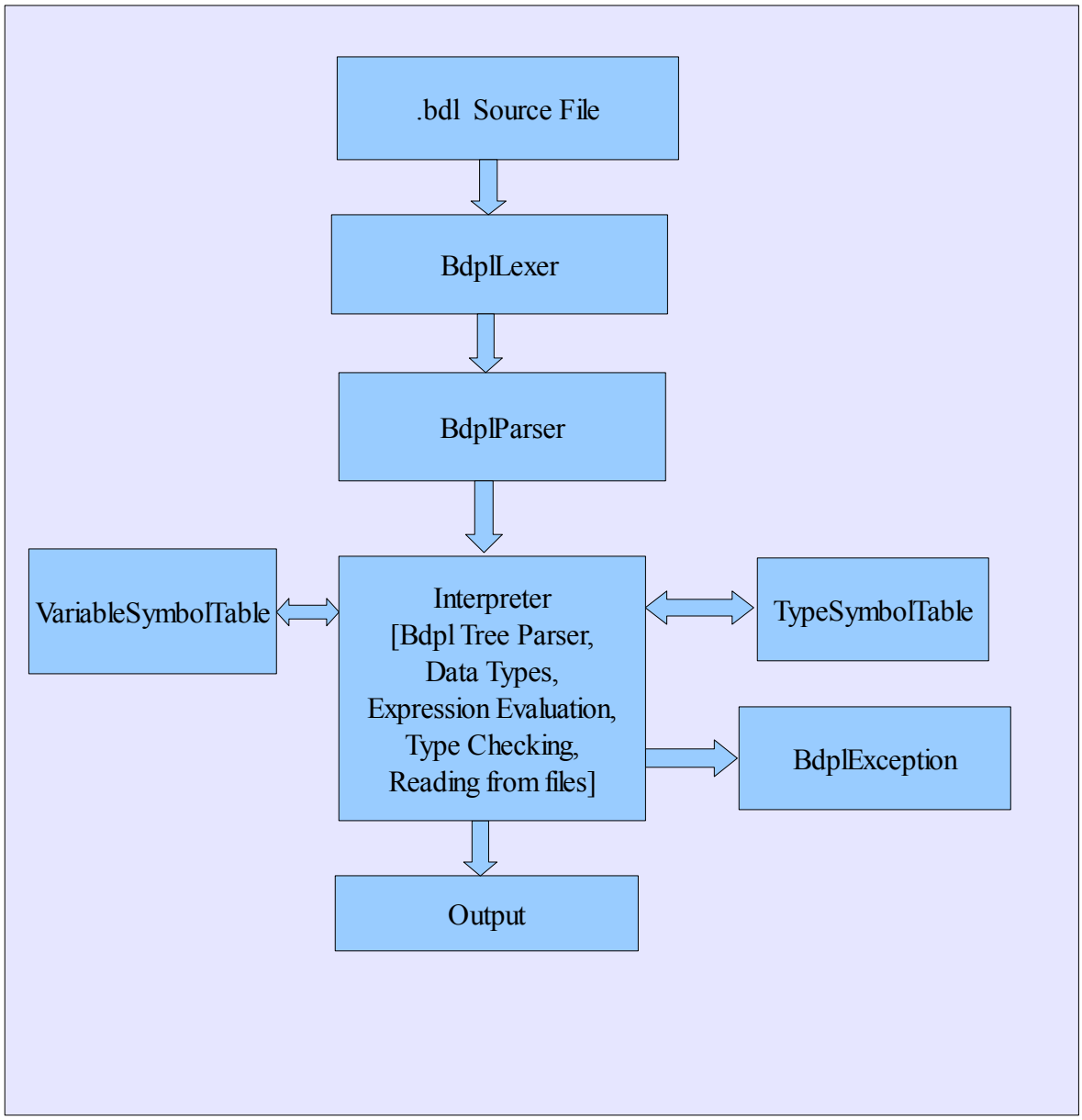
Automated JavaDoc and UML class diagram generation.

5 Architectural Design

BDPL is a programming language for processing and manipulating binary files. The design for the interpreter for BDPL involved two major parts. The development of the front end and the backend.

The front end consists of the lexer, parser and the AST walker. The backend consists of the implementation for data types, expression evaluation, type checking, type conversion and reading files into data types.

5.1 Block Diagram



The basic block diagram representing the high level structure of the Bdpl interpreter is as shown in the diagram. The lexer(BdplLexer) takes in the .bdl file as the input and translates a stream of characters into a stream of tokens and gives it to the parser. The parser(BdplParser) analyses the structure and checks whether it conforms to the grammar of our language. It then translates this into a parse tree. The interpretation of the program begins at this stage where the various data nodes are created as the tree is walked and the symbol tables are checked. Here expression evaluation, type checking, type conversion and reading of file s to be manipulated take place.

5.2 Description of Design

The design and implementation of the frontend which includes the lexer, parser and tree walker were done using ANTLR 2.0. The design for the backend involves the implementation of the data types, symbol tables, expression evaluation, type checking , type conversion and reading data from the file which was done using java (version 1.5) . The design for the backend involved generation of a framework for the important components. This involved the designing of interfaces which enabled the efficient implementation of the other constructs.

The representation of the data nodes in Bdpl can be visualized as that of a tree. Here every data node can either be a basic type or a derived type. A basic type contains either an int, bit or byte. A derived type can either be a “struct” or an “array”. The main issue to be dealt here was the representation of structs and arrays. So the visualization of the derived data types was done in the form “tree”.

Here each member of the structure was considered as a child of the struct node. So keeping track of the members of the structure became easy. Similarly, in case of arrays each member of the array was a child of the array node. In case of arrays however we have arrays of basic types and arrays of structs themselves. Irrespective of the type of arrays the same visualization holds good for arrays.

This representation of the derived data types as a tree solved most of the hurdles which would be faced otherwise. This helped in the representation of “*” arrays, nested structures and a number of other features such as implementation of fieldsize and read. Also this provided a uniform framework and a clean approach to dealing with the data types.

All processing associated with the data nodes can be done recursively where each data node does the required operations on all its children and this can be achieved by just walking the data node tree.

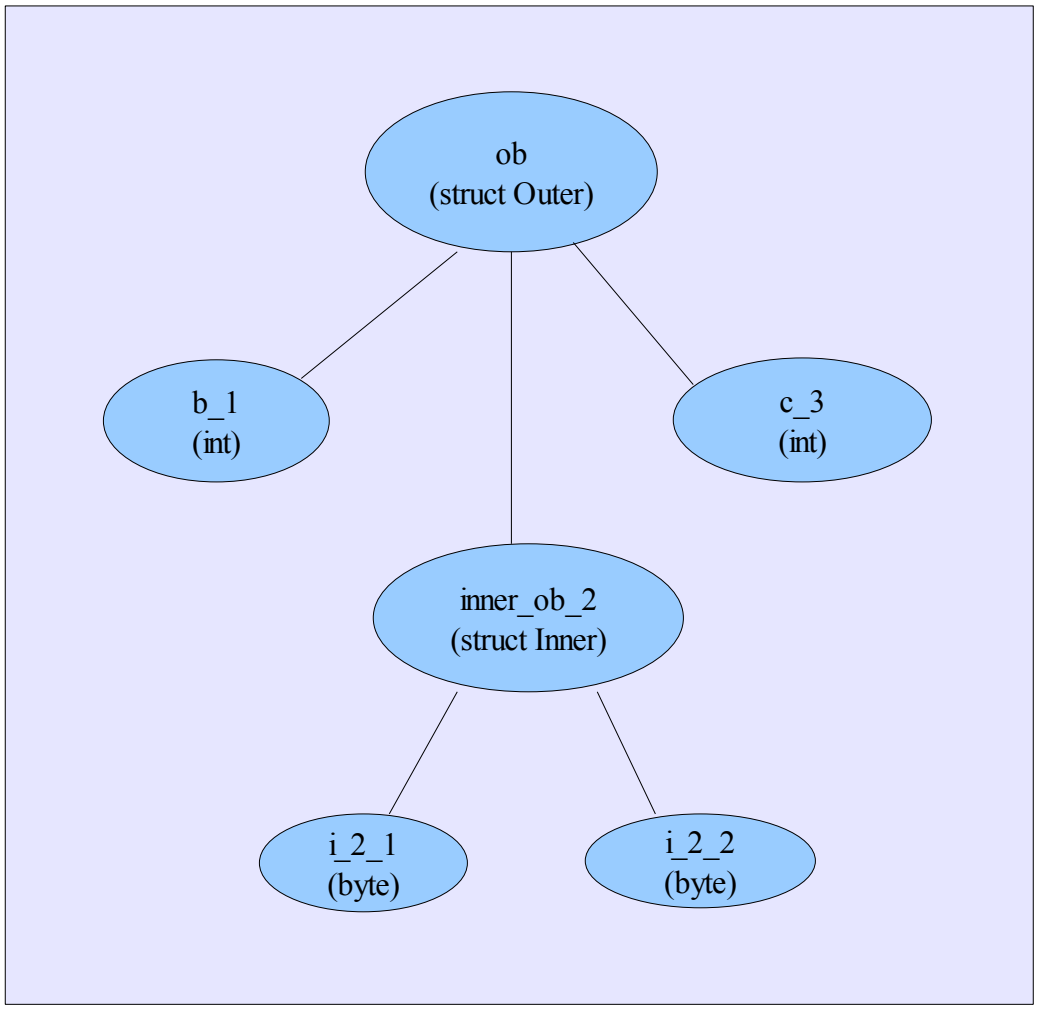
Some representation s of the structure s and the arrays are shown below.

Consider the following structure declaration:

```
struct Outer
{
      int b_1;
```

```
      struct Inner
      {
            byte i_2_1;
            byte i_2_2;
      } inner_ob_2;
      int c_3;
} ob;
```

The representation of this structure is viewed as:

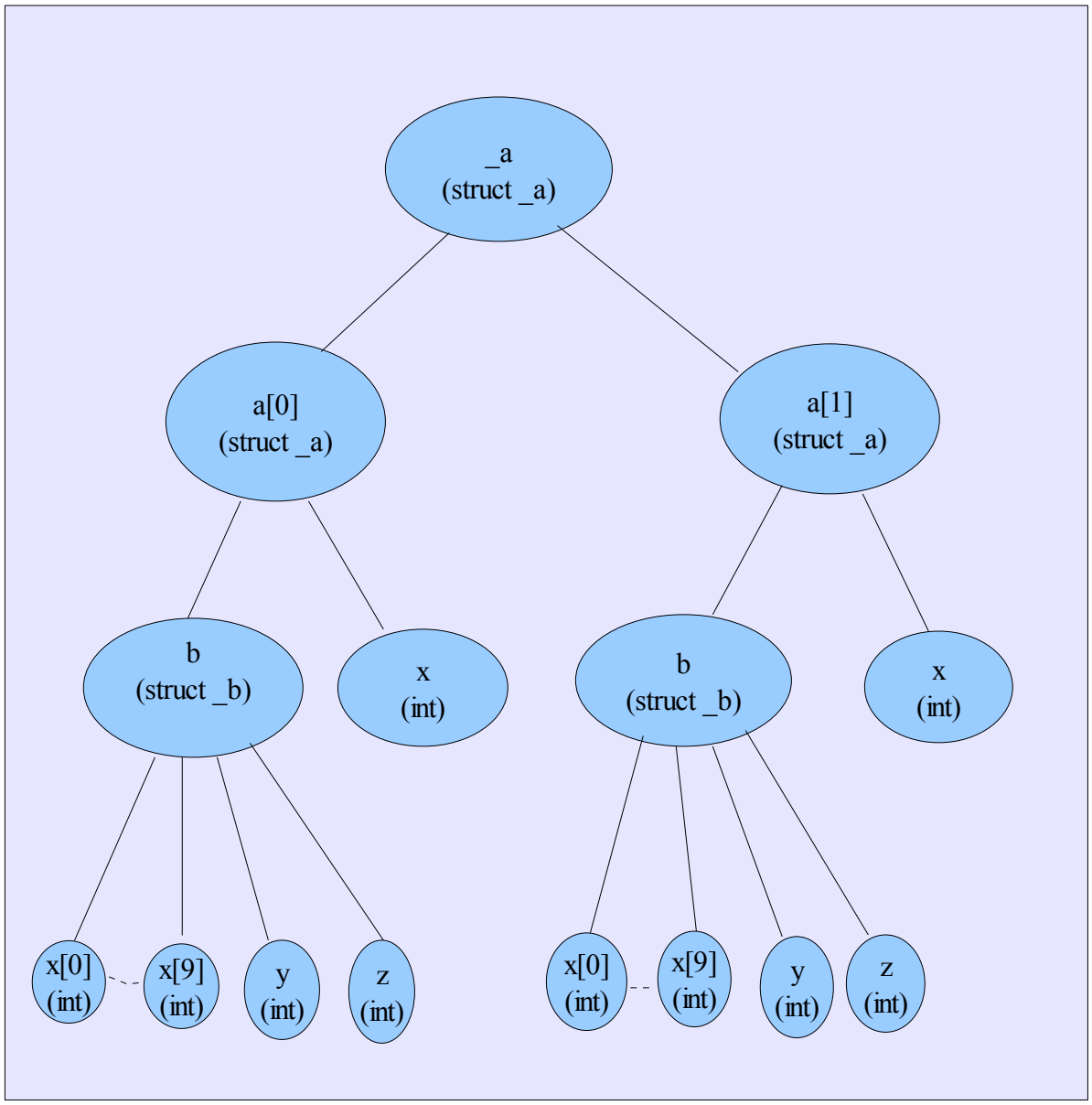


Consider the declaration of the array of structures as follows:

```
struct _a{
      struct _b
      {
            int[10] x;
            byte y;
            bit z;
```

```
      }b;
      int x;
}[2] a;
```

The representation of this array of structures is viewed as:

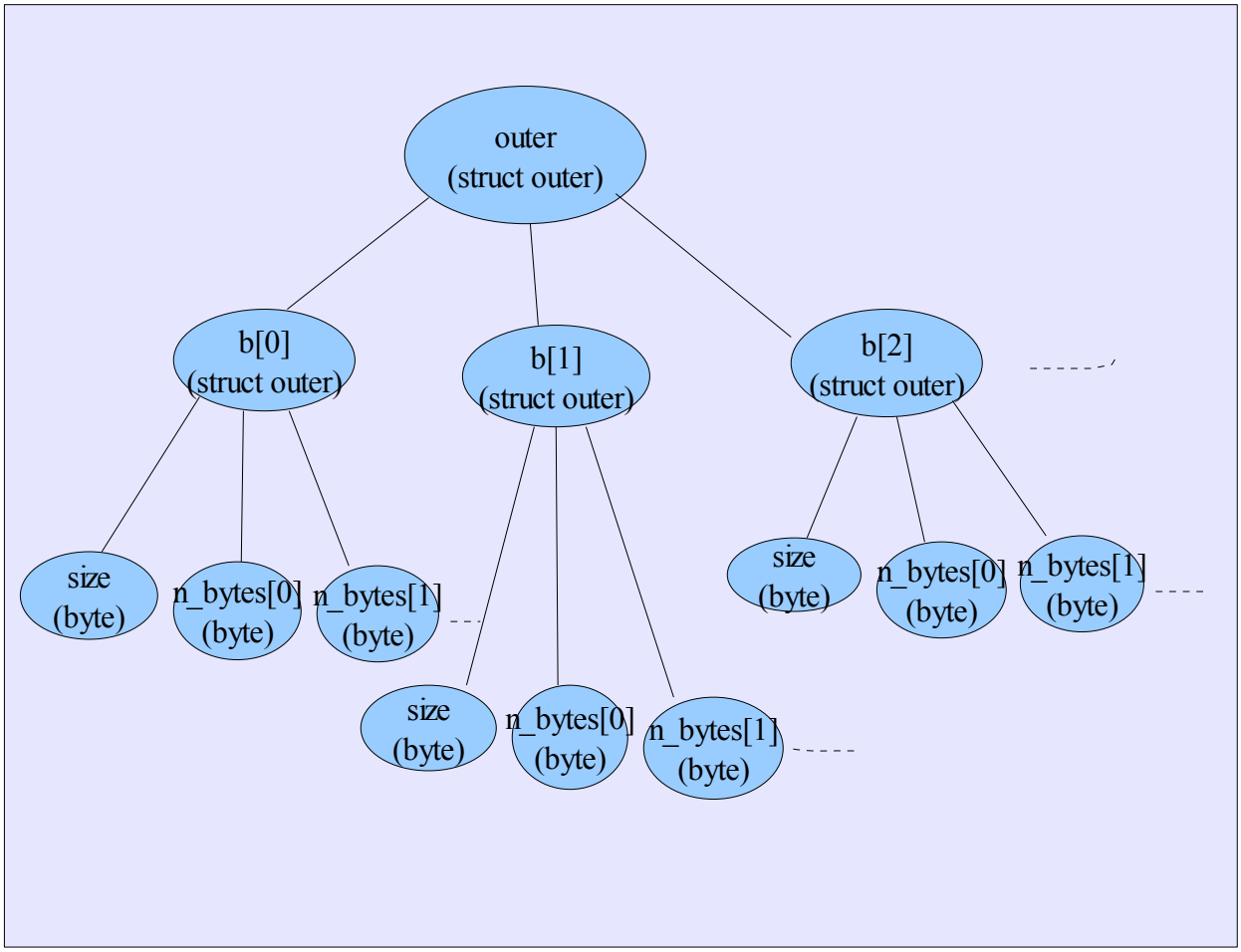


In case of "*" arrays, an example of the declaration is shown below:

```
struct outer
{
      byte size;
      byte[size] n_bytes;
```

```
}[*] b;
```

The representation as a tree is shown below:



Here the size of the array is considered unlimited, but the conceptual structure of the array is as shown. Also within the structure there is a byte array whose size is determined only during evaluation. This byte array is also represented similar to the unlimited size array. The dotted line indicates that the size is not known.

Using this as the framework the backend was developed. The design details of the parts in the frontend and backend are explained below.

5.2.1 The Lexer, Parser and Walker

The lexer, parser and Abstract Syntax Tree walker for the BDPL interpreter in written in Antlr.

The walker comprises a series of rules to descend through the tree, and actions associated with each rule, that either perform execution of a BDPL statement, or prepare data structures for BDPL data elements. The actions are written in the form of Java statements are the embedded in the antlr source code.

In BDPL, a program comprises declarations and statements. Every line of code that is written in a BDPL program is one of these. These statements may be written in any order, which means declarations are not required to be grouped at the start of the program source. They can be freely interspersed with executable statements, as long as all executable statements refer to variables that have been previously declared. The exceptions to this rule are statements that written within declarations, where variables may be used earlier in the text of the souce program than the text of the declarations for those variables. This is permitted because these statements are not executed when the declaration is parsed. They are only executed when the declared variable is used. This is explained in greater detail in the section on delayed expression evaluation. However, the following rule is held sacrosanct - Wherever an executable statement may be written, the variables used in it *must* be declared prior to the execution of that statement. Any violation of this rule will result in an error being reported by the static sematic analyser built into the walker.

The grouping of the AST nodes and sub-trees into categories is best indicated by the following tree diagram. The structure is representative of the hierarchy that the BDPL parser builds into the tree.

Declarations

A declaration is a subtree which has one of the following tokens or strings as a root node. Tokens are capitalized. Keywords are in lower case.

- ARRAY
- int
- byte
- bit
- type

Executable Statements

Statements are represented in the AST by subtrees which comprise one of the following keywords as root nodes. The handling of these statements by the walker is detailed in the sections to follow.

- if
- for
- read

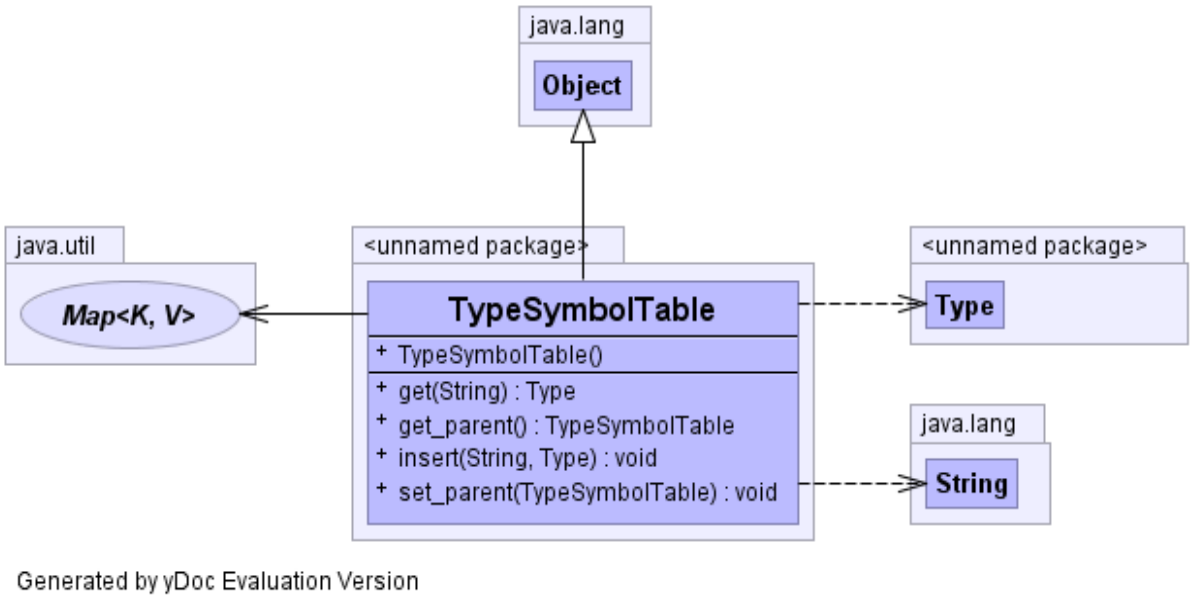
- write
- print
- printstring
- break
- continue

5.2.2 Scoping and Symbol Table

We had two symbol tables : The variable symbol table and the Type symbol table. The variable symbol table stores pointers to identifiers and the type symbol table stores pointers to Type classes.

Type Symbol tables and structures

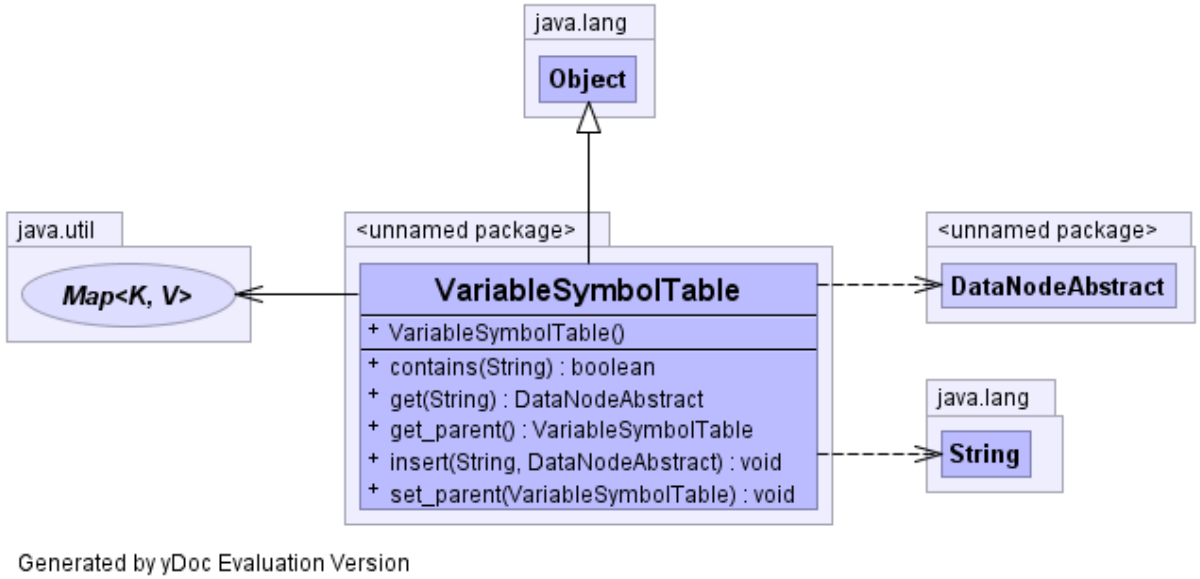
Since types can be defined inside structures (and they will be visible only inside the structure), each structure has its own type symbol table. The type symbol table itself can store a pointer to its parent, thus making the types in the parent accessible. We thus form a linked list of type symbol tables to encapsulate the scoping of types.



Variable symbol table

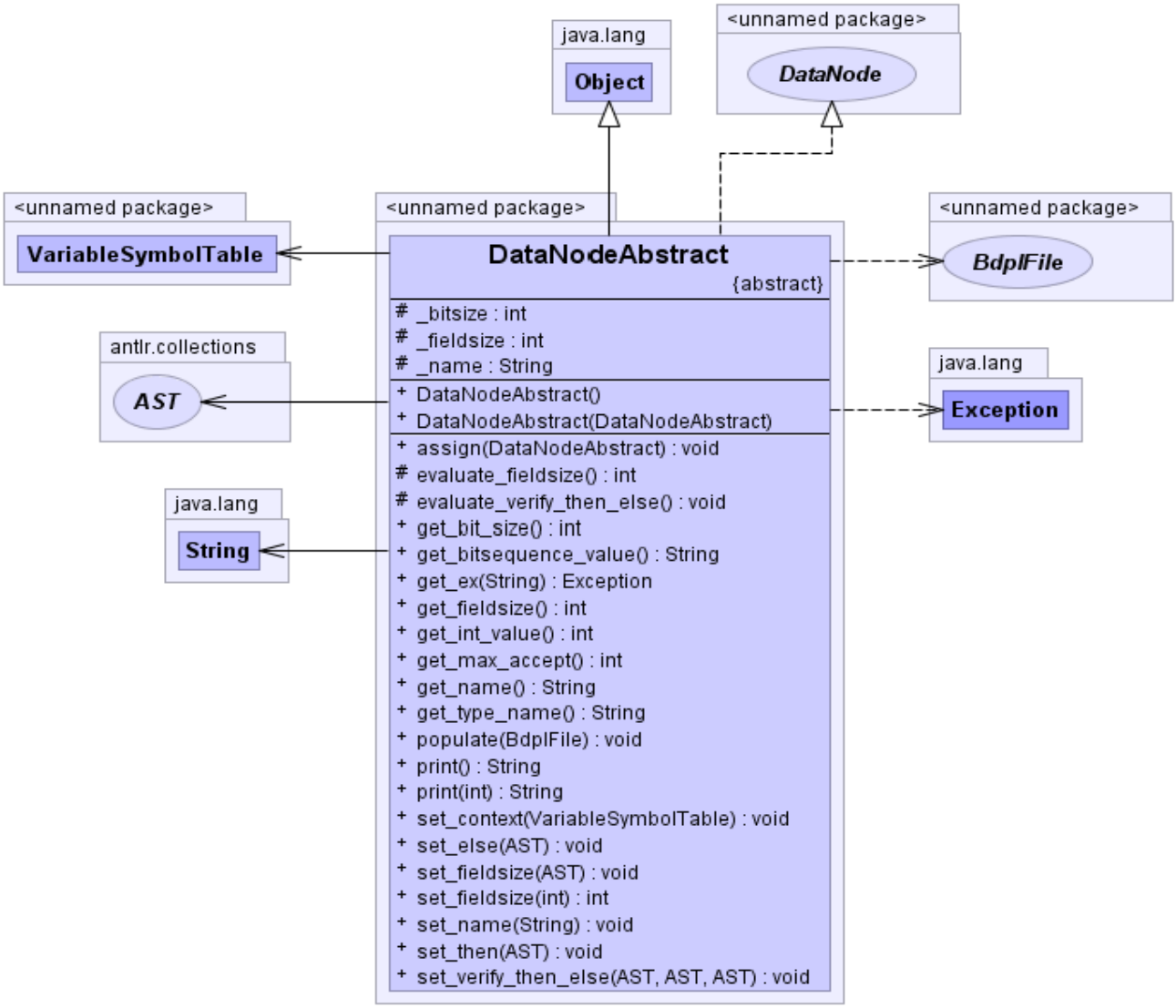
Bdpl has a challenging scoping scheme for variables. Every struct starts a new scope. Since we have arrays of structs, every element of an Array of structs has its own scope. Furthermore, every identifier can have

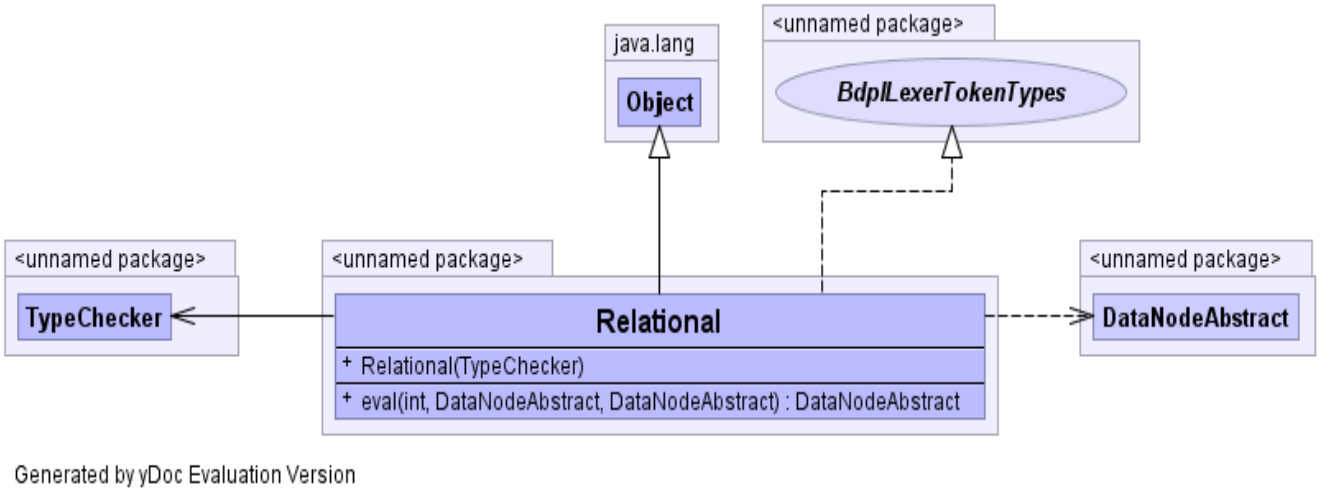
code associated with it which should be executed every time the identifier is populated. For enabling this, we store a pointer to the variable symbol table for every identifier. Every struct starts a new scope and hence there is a new variable symbol table defined for each struct. A linked list of symbol tables is maintained (just like the type symbol table) to encapsulate the scoping of variables.



5.2.3 Data Types

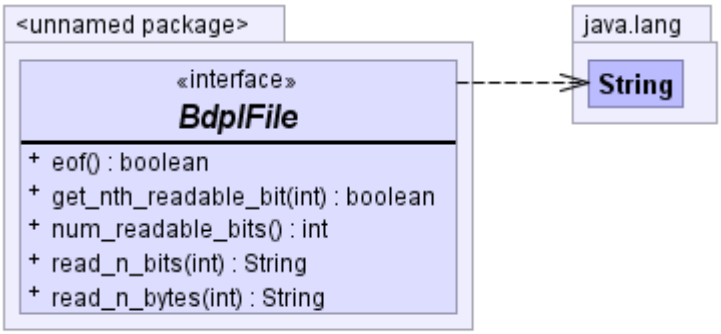
Interface based design was used for implementing nodes representing data. An interface was defined that all the data types nodes conformed to. A base class from which all nodes are derived was defined. The AST walker invoked the functionality of the data nodes thru this interface. Common parts were implemented in the base class (called DataNodeAnstract) and specific implementation was done in subclasses (one for each type, i.e bit,byte, int, struct, array).


java.lang
Object
<unnamed package>
DataNode
<unnamed package>
VariableSymbolTable
<unnamed package>
DataNodeAbstract
{abstract}
_bitsize : int
_fieldsize : int
_name : String
+ DataNodeAbstract()
+ DataNodeAbstract(DataNodeAbstract)
+ assign(DataNodeAbstract) : void
# evaluate_fieldsize() : int
# evaluate_verify_then_else() : void
+ get_bit_size() : int
+ get_bitsequence_value() : String
+ get_ex(String) : Exception
+ get_fieldsize() : int
+ get_int_value() : int
+ get_max_accept() : int
+ get_name() : String
+ get_type_name() : String
+ populate(BdplFile) : void
+ print() : String
+ print(int) : String
+ set_context(VariableSymbolTable) : void
+ set_else(AST) : void
+ set_fieldsize(AST) : void
+ set_fieldsize(int) : int
+ set_name(String) : void
+ set_then(AST) : void
+ set_verify_then_else(AST, AST, AST) : void
<unnamed package>
BdplFile
antlr.collections
AST
java.lang
Exception
java.lang
String



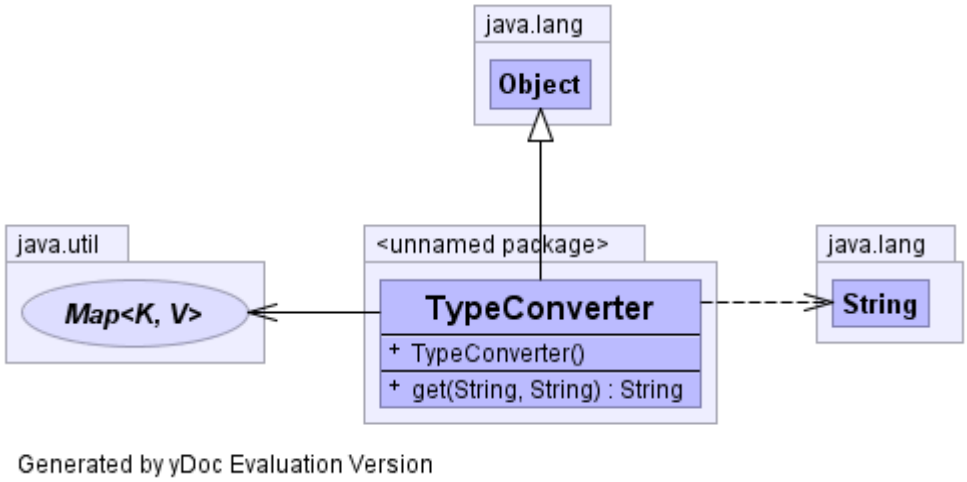
5.2.4 Reading from a File

Files were implemented using interfaces. An interface called BdplFile was defined which had 2 implementations : BdplDataFile and BdplMemFile . The BdplDataFile represented a file from disk It had a constructor which read the file form the disk and stored it in memory for use. The BdplMemFile on the other hand was directly created by giving it data. The interface defined operations for getting and testing data (in bytes or bits) from the file.

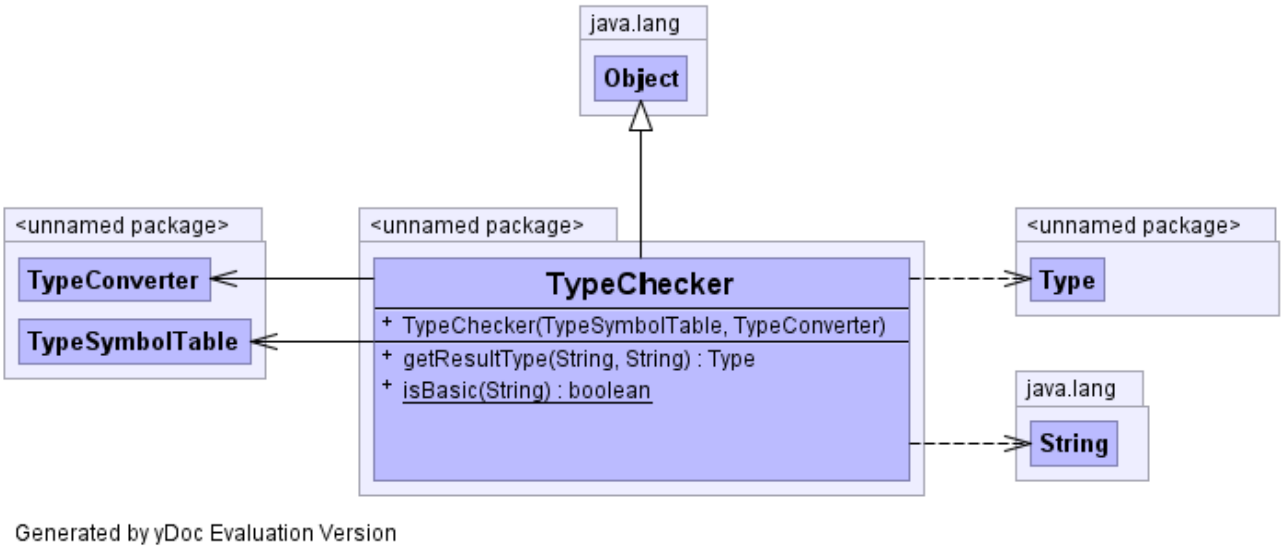


5.2.5 Type Conversion and Type Checking

Type classes were written for all types. Whenever a new type was defined (by defining a struct), an entry into the type symbol table of the current scope was made. The type symbol table stored a pointer to the Type object representing the type.



Type conversion was implemented in the DataNodes by using a function called assign that Assign accepts an object of type DataNodeAbstract. The actual type of this object was determined and and appropriate conversion was done. If conversion is not possible, an exception was thrown.



6 Testing

The project used a TCL script based automated test scheme for all testing. All BDPL programs generated output in the form of print statements that dump information to the console. This output is captured in a file and compared with a 'golden' version of the output which is certified to be correct. A section of the appendix at the end of this document offers a small sample of the diverse range of test programs that were used to ceritfy the interpreter. These include small test programs that test individual features as well as complete programs that work on live data from real world files and produce output. The first two programs in this list were demonstrated as classical examples of the power of BDPL.

The following is the TCL script used to run automated tests

```
set javaexec java

set env(CLASSPATH) {x:\tools\antlr.jar;x:\lib\bdpl.jar}
puts $env(CLASSPATH)

set testPath [file nativename [file join "." "test"]]
cd $testPath

# Collect all the bdl files that are in the current directory
set files [glob *.bdl]

# Loop through all the files that have been found
foreach f $files {
 set result [exec -- $javaexec BdplMain -f $f 2>&1 > $f.out]
 catch {set result [exec -- diff -b -q $f.out $f.golden ]} error
 switch $error "" "puts {Testing $f......OK}" "default" "puts {Testing $f .......FAILED $error}"
}
```

7 Lessons Learned

7.1 Akshay

This project didn't just teach me how to build a compiler and work in a team. It taught me how to think differently and build something new. The most enjoyable part of the project was conceptualizing the Bdpl language, making the constructs, and finally, making it work. It may be tough doing things a little different, but it sure is worth it.

I must stress on the importance of timing . Starting too early doesn't help, you'd be clueless. Start late and you wont have enough time. The best thing to do it to speak to your teacher and get ideas from him/her. Having a test suite early on in the process will prevent situations where time would be spent a lot of time fixing code.

I found google code to be awesome. They have an online bug tracking system in addition to a svn repository. Source code control was essential, but the bug tracking helped too. Project management doesn't really work. The only way to finish your stuff well in time would probably be to go full throttle at all times.

7.2 Bharadwaj

Good results in software can be obtained from a good architecture. A good design goes a long way in making a complex piece of software like a compiler come together with facility. This is exactly what could be learned from this project experience. Design for scalabilty and extensibility is critical.

Another learning: antlr is not the most convenient language in which to write a grammar. As novice users of grammar, we were required to grapple a lot with the tool to coerce it o generate trees in the form most suitable to the implementation of our language. While it did become easier with time, much of the grammar had to be re-written in a style acceptable to an LL parser.

One of our strongest points I believe was that we conceptualized the language without regard for how it would be implemented. We then chose the set of features and timelines for implementing them and implemented nearly all of the features we envisioned with nearly exactly the same syntax and semantic as when we designed the language. The learning to be derived from this is that it is important to have sound ideologies behind a language. Having a philosophy to a programming language gives programmers in the langauge the ability to extrapolate fundamental principles without having to bother about minute details and special cases. It also makes for clear thinking in what features the language should have and should not have, and goes a long way even in doing the implementation the right way.

7.3 Preethi

The implementation of this project helped me learn about the various stages involved in the design of an interpreter. Till now, I had just learnt about all these stages theoretically without actually having any hands on experience with the design. The implementation of this project gave me the opportunity to learn ANTLR . I also learnt a lot about team work and how important it is to be active as part of a team. Being part of a team project requires a lot of hard work and patience.

Another source of learning for me would be from my teammates. Working with them helped me learn a whole lot more than what I would have by myself.

My advice to future teams will be to stick to deadlines and schedules so that the whole process is smooth.

7.4 Aditi

This project was a wonderful learning experience. The key to our project was the sound framework which was made keeping the minute details in mind. This project gave me a totally new insight towards the binary world. “Early to Start,Early to Finish” should be the punch lines for such types of creative projects. It helped me in increasing my understanding of compilers. Working in team gained me the confidence to work with such critical projects where every small thing create a lot more difference. My teammates stood like pillers for the design of our language. I learned lot more working as a team.

8 Appendix

8.1 Source Code

```
/*
 * This is the BDPL ANTLR grammar file
 *
 */

////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                                    LEXER                                   //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
class BdplLexer extends Lexer;

options {
    charVocabulary = '\3'..'\377';
    k = 3;
}

protected
LETTER
    : ('a'..'z'|'A'..'Z')
    ;

protected
DIGIT
    : ('0'..'9')
    ;

protected
UNDERSCORE
    : '_'
    ;

SEMICOLON : ';';
TERMINATOR : ":-)";
COMMA : ',';
OPENBRACE : '{';
CLOSEBRACE : '}';
SQBROPEN : '[';
SQBRCLOSE : ']';
OPENPAREN : '(';
CLOSEPAREN : ')';
STAR : '*';
SLASH : '/';
ASSIGN : '=';
DOT_DOT : "..";
DOT : '.';
NEGATE : '~';
NOT : '!';
MINUS : '-' ;
PLUS : '+';
PERCENT : '%';
SET : "=>";
EQUALITY : "==";
INEQUALITY : "!=";
LLSH : "<<";
LRSH : ">>";
ALSH : "<<<";
ARSH : ">>>";
GT : ">";
LT : "<";
GTE : ">=";
LTE : "<=";
ROL : "<-<";
ROR : ">->";
```

```
QUESTION :'?';
APPEND : "<-";
INDEX : "$#";
BYTEOFFSET : '$';
POUND : '#';
LAND : "&&";
BAND : '&';
BIOR : '|';
BEOR : '^';
LOR : "||";

ID
    : (LETTER | UNDERSCORE)(LETTER|DIGIT|'_')*
    ;

protected
BINPREFIX
    : '0''b'
    ;

protected
BINDIGITS
    : ('0'|'1')
    ;

protected
BINNUM
    : BINPREFIX(BINDIGITS)+
    ;

protected
DECNUM
    : (DIGIT)+
    ;

protected
HEXDIGITS
    : ('0'..'9' | 'a'..'f' | 'A'..'F')
    ;

protected
HEXPREFIX
    : '0'('x'|'X')
    ;

protected
HEXNUM
    : HEXPREFIX (HEXDIGITS)+
    ;

NUM
    : DECNUM
    | HEXNUM
    | BINNUM
    ;

WS
    : (' '
    | '\n' {newline();}
    | ('\r' '\n') => '\r' '\n' {newline();}
    | '\r' {newline();}
    | '\t'
    ) {$setType(Token.SKIP);}
    ;

STRING
    : '"'! ( '"' '"'! | ~('"'))* '"'!
    ;

COMMENT
    : '/' '/' ((~'\n')* '\n' {newline();}) {$setType(Token.SKIP);}
```

```
    ;

////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                                    PARSER                                  //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
{import java.util.*; }
class BdplParser extends Parser;

options{
    k = 2;
    buildAST = true;
}

tokens{
    ARRAY;
    BODY;
    BLOCK;
    IDEN;
    INITLIST;
    OPTIONAL;
    PROG;
    RANGES;
    TAG;
    SATISFIES;
    INITIAL;
    NULL;
    COND;
    INCR;
    ARRAY_SIZE;
    DECL;
       LVALUE;
    STRUCT_NAME;
}

program

    : (stmt)+ EOF!
        {#program = #([PROG,"PROG"], #program);}
    ;

stmt
    : declstmt
    | execstmt
    ;

expr
    : (term ASSIGN^ tern_expr) => assign_expr
    | (term APPEND^ tern_expr) => append_expr
    | tern_expr
    ;

append_expr
    : term APPEND^ tern_expr
    ;

assign_expr
    : term ASSIGN^ tern_expr
    ;

execstmt
    : expr SEMICOLON!
    | stmtblock
    | "if"^ "("! expr ")"! execstmt (options {greedy=true;}: "else"! execstmt)?
    | "for"^ "("! (init_iterator)? SEMICOLON! (cond_iterator)? SEMICOLON! (incr_iterator)? ")"!
execstmt
    | "read"^ "("! (ID) COMMA! expr ")"! SEMICOLON!
    | "write"^ "("! (ID) COMMA! ID ")"! SEMICOLON!
    | "set"^ "("! STRING "=>"! STRING COMMA! ID ")"! SEMICOLON!
    | "exit"^ "("! STRING ")"! SEMICOLON!
```

```
    | "print"^ "("! (STRING|expr) ")"! SEMICOLON!
    | "printstring"^ "("! (expr) ")"! SEMICOLON!
    | "break" SEMICOLON!
    | "continue" SEMICOLON!
    ;

init_iterator
    : expr
        {#init_iterator = #([INITIAL,"INITIAL"],#init_iterator);}
    ;

cond_iterator
    : expr
        {#cond_iterator = #([COND,"COND"],#cond_iterator);}
    ;

incr_iterator
    : expr
        {#incr_iterator = #([INCR,"INCR"],#incr_iterator);}
    ;

stmtblock
    : "{"! (execstmt)* "}"!
        {#stmtblock = #([BLOCK,"BLOCK"],#stmtblock);}
    ;

declstmt
    :   (   "bit"^
            | "byte"^
            | "int"^
            | "float"^
            | "double"^
            | ("type"^ "struct"! ID)
            | (("struct"^) tag struct_body)
        )
        ( {#declstmt=#([ARRAY,"ARRAY"],declstmt);} array_defn )?
        list_of_ids
        (valid_check)?
        (optional_check)?
        SEMICOLON!
    | "file"^ STRING ID SEMICOLON!
    ;

tag
    : ID
        {#tag = #([TAG,"TAG"],#tag);}
    ;

array_defn
    : (SQBROPEN! (expr|STAR) SQBRCLOSE!)
        {#array_defn = #([ARRAY_SIZE,"ARRAY_SIZE"],#array_defn);}
    ;

struct_body
    : ("{"! (declstmt)* "}"!)
        {#struct_body = #([BODY,"BODY"],#struct_body);}
    ;

range_list
    : (range_element)(COMMA! range_element)*
        {#range_list = #([RANGES,"RANGES"],#range_list);}
    ;

range_element
    : (expr)(DOT_DOT^ expr)?
    ;

list_of_ids
    : (single_id) //(COMMA! single_id)*
    ;
```

```
single_id
    : ID ("("! init_list ")"!)? (size)?
        {#single_id = #([IDEN,"IDEN"],#single_id);}
    ;

size
    : ("("! "fieldsize"^ expr ")"!)
    ;

valid_check
    : valid_values (ok_block)? (nok_block)?
        {#valid_check = #([SATISFIES,"SATISFIES"],#valid_check);}
    ;

valid_values
    : "satisfies"! ("{"! expr "}"!)
    ;

ok_block
    : ("then"^ stmtblock)
    ;

nok_block
    : ("else"^ stmtblock)
    ;

optional_check
    : "optional"! "on"! expr
        {#optional_check = #([OPTIONAL,"OPTIONAL"],#optional_check);}
    ;

init_list
    : (initializer)(COMMA! initializer)*
        {#init_list = #([INITLIST,"INITLIST"],#init_list);}
    ;

initializer
    : (STRING "=>"^ STRING)
    ;

tern_expr
    : lor_expr(QUESTION tern_expr COLON tern_expr)?
    ;

lor_expr
    : and_expr (LOR^ and_expr)*
    ;

and_expr
    : bior_expr (LAND^ bior_expr)*
    ;

bior_expr
    : beor_expr (BIOR^ beor_expr)*
    ;

beor_expr
    : band_expr (BEOR^ band_expr)*
    ;

band_expr
    : eq_expr (BAND^ eq_expr)*
    ;

eq_expr
    : comp_expr ((EQUALITY^ | INEQUALITY^) comp_expr)*
    ;

comp_expr
    : sh_expr ((GT^ | LT^ | GTE^ | LTE^) sh_expr)*
    ;
```

```
sh_expr
    : sum_expr ((LLSH^ | LRSH^ | ALSH^ | ARSH^ | ROL^ | ROR^) sum_expr)*
    ;

sum_expr
    : mult_expr ((PLUS^ | MINUS^) mult_expr)*
    ;

mult_expr
    : not_expr ((STAR^ | SLASH^ | PERCENT^) not_expr)*
    ;

not_expr
    : ((NEGATE^ | NOT^)? child_term)
    ;

child_term
    : term
    | "("! expr ")"!
    | "$#"^ lvalue
    | POUND^ lvalue_term
    | NUM
    ;

term
    : lvalue
    | BYTEOFFSET^ lvalue_term
    ;

lvalue
    : lvalue_term (DOT! lvalue_term)*
        {#lvalue = #([LVALUE,"LVALUE"],#lvalue);}
       ;

lvalue_term
    : object(SQBROPEN^ (tern_expr) SQBRCLOSE!)?
    ;

object
       : ID
       ;

////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//                                TREE WALKER                                 //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
{
       import java.util.regex.Pattern;
}
//
// A program is a collection of declarations and statements
// A declaration is a subtree which has one of the following as a root node
//   ARRAY
//   int
//   byte
//   bit
//   type
// A statement may either be an expression or one of the following
//   if
//   for
//   read
//   write
//   set
//   print
//   break
//   continue
// For each of the above, the keyword itself forms the root of the
// subtree that constitutes the statement.
// A program returns nothing
```

```
// A statement returns nothing
// An expression returns an AbstractDataNode
//
class BdplTreeParser extends TreeParser;
{
    public VariableSymbolTable varSymbTbl = new VariableSymbolTable();
    TypeSymbolTable typeSymbTbl = new TypeSymbolTable();
    TypeConverter typeConverter = new TypeConverter();
    TypeChecker typeChecker = new TypeChecker(typeSymbTbl,typeConverter);
    Arithmetic arith = new Arithmetic(typeChecker);
    Relational relate = new Relational(typeChecker);
    Logical logic = new Logical(typeChecker);
    Bitwise bitwise = new Bitwise(typeChecker);
    DataNodeAbstract r;
    boolean breakset = false;
    boolean continueset = false;
    int loopcounter = 0;
    BdplFile inputFile;
}

program throws Exception
{
}
    : #(PROG ((stmts
    | (r=decls)
        {
            if(r==null)
            {
                //System.out.println(" r is null !! \n");
            }
            else
            {
                varSymbTbl.insert(r.get_name(),r);
            }
        }

     )* // decls or stmnt
    )

    )
    ;

stmts throws Exception
{
    DataNodeAbstract r;
    DataNodeAbstract init;
    String source;
    DataNodeAbstract dest;
    if(breakset || continueset) return;
}
    : #("if" r=expr thenpart:. (elsepart:.)?
        {
            //
            // VRB: To do
            // Check the type of 'r' for consistency
            //

                if(r.get_int_value() > 0){
                     stmts(#thenpart);
                }else{
                    if(null != elsepart){
                        stmts(#elsepart);
                    }else{
                        //
                        // Do nothing here. The else part is
                        // optional
                        //
                    }
                }

        }
```

```
       )
    | #("for" (#(INITIAL init=expr))? (#(COND cond:.))? (#(INCR incr:.))? (#(body:BLOCK {}))
       {
          loopcounter++;
          while(true){
              if(#cond != null && expr(#cond).get_int_value()== 0) break;
              stmts(#body);
              if(breakset) {breakset = false; break;}
              if(#incr != null) expr(#incr);
              if(continueset) {continueset = false; continue;}
          }
          loopcounter--;
       }
       )
    | #("break"                     {if(loopcounter != 0){
                                        breakset = true;
                                     }else{
                                        throw new BdplException("'break' is only permitted within the
body of a loop");
                                     }})
    | #("continue"                  {if(loopcounter != 0){
                                        continueset = true;
                                     }else{
                                        throw new BdplException("'continue' is only permitted within
the body of a loop");
                                     }})
    | #("read" source=id dest=expr  {
//                                      inputFile = fileTable.get(source);
                                        dest.populate(inputFile);
      })
    | #("write"                     {})
    | #("set"                       {})
    | #("print" {String str;}
                      (((str = string) {
                             int i;
                             String[] strings = str.split("::",-1);
                             for(i=0;i<strings.length-1;i++) {
                                    System.out.println(strings[i]);
                             }
                             System.out.print(strings[i]);
                             }
                      )
                |     (r = expr {System.out.print(r.print());}))
               )
    | #("printstring" ((r = expr) {System.out.print(r.print(1));}))
    | #(BLOCK (stmts)*              {})
    | r=expr                        {}
    ;

//
// Check the symbol table for this name
// If name exists, throw an exception
// Else, get a node of type int
// Populate the node with INITLIST info (AST)
// Populate the node with fieldsize info (AST)
//
decls returns [DataNodeAbstract r=null] throws Exception
{
    String name;
    String id;
}
    : #("file" name=string id=id {inputFile = new BdplDataFile(name);})
    | #(ARRAY  (type:.) (#(ARRAY_SIZE array_size:.))
        (#(IDEN id=id
        {
            DataNodeAbstract dummy_node=null;
            if(type.getText().equals("int") || type.getText().equals("bit") ||
type.getText().equals("byte"))
            {
                dummy_node=typeSymbTbl.get(type.getText()).getDataNode();
            }
```

```
        else
        if(type.getText().startsWith("struct") || type.getText().startsWith("type") )
        {
            VariableSymbolTable newST=new VariableSymbolTable();
            newST.set_parent(varSymbTbl);
            varSymbTbl=newST; // go into new scope

            dummy_node=decls(#type);
            //((DataNodeStruct)dummy_node).set_scope(varSymbTbl,typeSymbTbl);
            varSymbTbl=varSymbTbl.get_parent();
        }
        DataNodeArray arr;
        if(array_size.getText().equals("*"))
        {
            arr=new DataNodeArray(dummy_node);
            arr.set_is_unlimited(true);
            arr.set_scope(varSymbTbl);
        }
        else
        {
            arr = new DataNodeArray(dummy_node,array_size);
            arr.set_is_unlimited(false);
            arr.set_scope(varSymbTbl);
        }
        r=arr;
        r.set_name(id);
    }
    (#(INITLIST {}))?
    (#("fieldsize" f_ast_a:.
        {
            r.set_context(varSymbTbl);
            r.set_fieldsize(f_ast_a);
        }

    ))?) ) //iden ends
    (#(SATISFIES valid_a:.
        {
            r.set_context(varSymbTbl);
            r.set_verify_then_else(valid_a,null,null);
        }
        (#("then" then_ast_a:.
            {
                r.set_then(then_ast_a);
            }
         ))?
        (#("else" else_ast_a:.
            {
                r.set_else(else_ast_a);
            }

         ))?))?
    )

| #("int"
    (#(IDEN name=id
        {

            Type intType = typeSymbTbl.get("int");
            DataNodeInt intNode = (DataNodeInt)intType.getDataNode();
            intNode.set_name(name);
            r=intNode;
        }

    (#(INITLIST {}))?
    (#("fieldsize" f_ast_i:.
        {
            r.set_context(varSymbTbl);
            r.set_fieldsize(f_ast_i);
        }
    ))?)
```

```
    )+
    (#(SATISFIES valid_i:.
        {
            r.set_context(varSymbTbl);
            r.set_verify_then_else(valid_i,null,null);
        }
        (#("then" then_ast_i:.
            {
                r.set_then(then_ast_i);
            }
         ))?
        (#("else" else_ast_i:.
            {
                r.set_else(else_ast_i);
            }

         ))?))?
   )
| #("byte"
    (#(IDEN name=id
        {
            Type byteType = typeSymbTbl.get("byte");
            DataNodeByte byteNode = (DataNodeByte)byteType.getDataNode();
            byteNode.set_name(name);
            r=byteNode;
        }

    (#(INITLIST {}))?
    (#("fieldsize" f_ast_B:.
        {
            r.set_context(varSymbTbl);
            r.set_fieldsize(f_ast_B);
        }
    ))?
    )
    )+
    (#(SATISFIES valid_B:.
        {
            r.set_context(varSymbTbl);
            r.set_verify_then_else(valid_B,null,null);
        }
        (#("then" then_ast_B:.
            {
                r.set_then(then_ast_B);
            }
         ))?
        (#("else" else_ast_B:.
            {
                r.set_else(else_ast_B);
            }

         ))?))?
    )

| #("bit"
     (#(IDEN name=id
        {
            Type bitType = typeSymbTbl.get("bit");
            DataNodeBit bitNode = (DataNodeBit)bitType.getDataNode();
            bitNode.set_name(name);
            r=bitNode;
        }

     (#(INITLIST {}))?
     (#("fieldsize" f_ast_b:.
        {
            r.set_context(varSymbTbl);
            r.set_fieldsize(f_ast_b);
        }

    ))?)
```

```
    (#(SATISFIES valid_b:.
        {
            r.set_context(varSymbTbl);
            r.set_verify_then_else(valid_b,null,null);
        }
        (#("then" then_ast_b:.
            {
                r.set_then(then_ast_b);
            }
         ))?
        (#("else" else_ast_b:.
            {
                r.set_else(else_ast_b);
            }

         ))?))?

   )+
   )
| #("struct"  (#(TAG name=id)) (body:.)
    {
        VariableSymbolTable newST=new VariableSymbolTable();
        TypeSymbolTable newTT=new TypeSymbolTable();
        newST.set_parent(varSymbTbl);
        newTT.set_parent(typeSymbTbl);
        varSymbTbl=newST; // go into new scope
        typeSymbTbl=newTT; // new type symbol table for type definitions
        DataNodeStruct structNode = new DataNodeStruct();
        structNode.set_scope(varSymbTbl,typeSymbTbl);
        AST child=body.getFirstChild();
        while(child!=null)
        {

            DataNodeAbstract cdn=decls(child);
            if(cdn.getClass().getCanonicalName ().equals ("DataNodeArray"))
            {
                ((DataNodeArray)cdn).set_scope(varSymbTbl);
            }
            structNode.set_child_by_name(cdn.get_name(),cdn);
            varSymbTbl.insert(cdn.get_name(),cdn);
            child=child.getNextSibling();

        }
        structNode.set_type_name("struct:"+name);
        r=structNode;

        Type structType=new Type ("struct:"+name,body);

        //insert this defn in the parent type st , and restore values
        typeSymbTbl=typeSymbTbl.get_parent();
        typeSymbTbl.insert("struct:"+name, structType);
        varSymbTbl=varSymbTbl.get_parent(); // go up a scope

    }
    (#(IDEN id=id
        {
            r.set_name(id);
        }

    (#(INITLIST {}))?
    (#("fieldsize" f_ast_s:.{}))?)
        {
            r.set_context(varSymbTbl);
            r.set_fieldsize(f_ast_s);
        }

    )?
    (#(SATISFIES valid_s:.
        {
            r.set_context(varSymbTbl);
            r.set_verify_then_else(valid_s,null,null);
```

```
        }
        (#("then" then_ast_s:.
            {
                r.set_then(then_ast_s);
            }
         ))?
        (#("else" else_ast_s:.
            {
                r.set_else(else_ast_s);
            }

         ))?))?
    )
|   #("type" name=id
    {
        Type t=typeSymbTbl.get("struct:"+name);
        AST child=t.get_ast().getFirstChild();
        DataNodeStruct structNode = new DataNodeStruct();
        structNode.set_type_name("struct:"+name);

        VariableSymbolTable newST=new VariableSymbolTable();
        TypeSymbolTable newTT=new TypeSymbolTable();
        newST.set_parent(varSymbTbl);
        newTT.set_parent(typeSymbTbl);

        varSymbTbl=newST;
        typeSymbTbl=newTT;


        structNode.set_scope(varSymbTbl,typeSymbTbl);

        while(child!=null)
        {
            DataNodeAbstract cdn=decls(child);
            structNode.set_child_by_name(cdn.get_name(),cdn);
            child=child.getNextSibling();
        }


         r=structNode;
         typeSymbTbl=typeSymbTbl.get_parent();
         varSymbTbl=varSymbTbl.get_parent();
     }
     ((#(IDEN name=id
         {
             r.set_name(name);
         }
     (#(INITLIST {}))?
     (#("fieldsize" f_ast_t:.
         {
             r.set_context(varSymbTbl);
             r.set_fieldsize(f_ast_t);
         }
     ))?))?
    )
     (#(SATISFIES valid_t:.
         {
             r.set_context(varSymbTbl);
             r.set_verify_then_else(valid_t,null,null);
         }
         (#("then" then_ast_t:.
             {
                 r.set_then(then_ast_t);
             }
          ))?
         (#("else" else_ast_t:.
             {
                 r.set_else(else_ast_t);
             }

          ))?))?
```

```
        )
    ;

expr returns [DataNodeAbstract r] throws Exception
{
    DataNodeAbstract a,b;
    r = new DataNodeBit();
    String y,z;
}
    : #(BYTEOFFSET  a=expr        {})
    | #(DOT         a=expr b=expr {})
    | #(PLUS        a=expr b=expr {r = arith.eval(PLUS,a,b);})
    | #(MINUS       a=expr b=expr {r = arith.eval(MINUS,a,b);})
    | #(STAR        a=expr b=expr {r = arith.eval(STAR,a,b);})
    | #(SLASH       a=expr b=expr {r = arith.eval(SLASH,a,b);})
    | #(PERCENT     a=expr b=expr {r = arith.eval(PERCENT,a,b);})
    | #(LLSH        a=expr b=expr {r = bitwise.eval(LLSH,a,b);})
    | #(LRSH        a=expr b=expr {r = bitwise.eval(LRSH,a,b);})
    | #(ALSH        a=expr b=expr {r = bitwise.eval(ALSH,a,b);})
    | #(ARSH        a=expr b=expr {r = bitwise.eval(ARSH,a,b);})
    | #(GT          a=expr b=expr {r = relate.eval(GT,a,b);})
    | #(LT          a=expr b=expr {r = relate.eval(LT,a,b);})
    | #(GTE         a=expr b=expr {r = relate.eval(GTE,a,b);})
    | #(LTE         a=expr b=expr {r = relate.eval(LTE,a,b);})
    | #(EQUALITY    a=expr b=expr {r = relate.eval(EQUALITY,a,b);})
    | #(INEQUALITY  a=expr b=expr {r = relate.eval(INEQUALITY,a,b);})
    | #(ROL         a=expr b=expr {r = bitwise.eval(ROL,a,b);})
    | #(ROR         a=expr b=expr {r = bitwise.eval(ROR,a,b);})
    | #(LAND        a=expr c:.         {r = logic.shortckt(LAND,a)?a:logic.eval(LAND,a,expr(#c));})
    | #(LOR         a=expr d:.    {r = logic.shortckt(LOR,a)?a:logic.eval(LOR,a,expr(#d));})
    | #(BAND        a=expr b=expr {r = bitwise.eval(BAND,a,b);})
    | #(BIOR        a=expr b=expr {r = bitwise.eval(BIOR,a,b);})
    | #(BEOR        a=expr b=expr {r = bitwise.eval(BEOR,a,b);})
    | #(APPEND      a=expr b=expr {})
       | #("$#"        arr:LVALUE {
                                          r = expr(#arr);
                                          if(r instanceof DataNodeArray){
                                                 r = new
DataNodeInt(((DataNodeArray)r).get_size());
                                          }else{
                                                 throw new BdplException("Size can only be applied
on an array");
                                          }
                                    })
    | #(ASSIGN      lhs:LVALUE b=expr {r = expr(#lhs); r.assign(b);})
    | #("=>"        y=string z=string {})
    | #(lval:LVALUE {}
        (
            (y=id {
                    r = varSymbTbl.get(y);
                }|
                (#(SQBROPEN y=id a=expr){
                  r = varSymbTbl.get(y);
                  if(r instanceof DataNodeArray){
                    r = ((DataNodeArray)r).get_element(a.get_int_value());
                  }
                })
            )
            (y=id {
                    if(r instanceof DataNodeStruct){
                        r = ((DataNodeStruct)r).get_child_by_name(y);
                    }else{
                        throw new BdplException(r.get_name()+" is not a structure");
                    }
                } |
                (#(SQBROPEN y=id a=expr){
                    if(r instanceof DataNodeStruct){
                        r = ((DataNodeStruct)r).get_child_by_name(y);
                        if(r instanceof DataNodeArray){
                            r = ((DataNodeArray)r).get_element(a.get_int_value());
```

```
                        }else{
                            throw new BdplException(r.get_name()+" is not an array");
                        }
                    }else{
                        throw new BdplException(r.get_name()+" is not a structure");
                    }
                })
            )*
        ){})
    | #(num:NUM     {r=new DataNodeInt(Integer.decode(num.getText()));})
    ;

protected
id returns [String r]
    {
        r = new String("");
    }
    :#(id:ID {r=id.getText();})
    ;

protected
num returns [int n]
{
    n = 0;
}
    :#(num:NUM {n=Integer.parseInt(num.getText());})
    ;

protected
string returns [String r]
{
    r = new String("");
}
    :#(str:STRING {r=str.getText();})
    ;
```

```
/*
 * Arithmetic.java
 *
 */

/**
 *
 * @author Bharadwaj Vellore
 */
public class Arithmetic implements BdplLexerTokenTypes{

    private TypeChecker _typeChecker;
    /** Creates a new instance of Arithmetic */
    //
    // To Do: Apply singleton pattern here
    //
    public Arithmetic(TypeChecker tc){
        _typeChecker = tc;
    }

    public DataNodeAbstract eval(
            int operator,
            DataNodeAbstract operand1,
            DataNodeAbstract operand2) throws Exception
    {
        DataNodeAbstract result;
        int res;

        if(!TypeChecker.isBasic(operand1.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }
        if(!TypeChecker.isBasic(operand2.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }

        Type resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand2.get_type_name());
        result = resultType.getDataNode();

        switch(operator){
            case PLUS:
                res = operand1.get_int_value() + operand2.get_int_value();
                break;
            case MINUS:
                res = operand1.get_int_value() - operand2.get_int_value();
                break;
            case STAR:
                res = operand1.get_int_value() * operand2.get_int_value();
                break;
            case SLASH:
                res = operand1.get_int_value() / operand2.get_int_value();
                break;
            case PERCENT:
                res = operand1.get_int_value() % operand2.get_int_value();
                break;
            default:
                res = 0;
                //
                // This point will never be reached
                //
                assert(false);
        }

        if(result instanceof DataNodeInt){
            return new DataNodeInt(res);
        }else if(result instanceof DataNodeByte){
            return new DataNodeByte(res);
        }else if(result instanceof DataNodeBit){
            return new DataNodeBit(res);
        }else{
            //
```

```
            // This should never happen
            //
            assert(false);
            return null;
        }
    }
}
```

```
import java.io.*;
import java.util.*;

/**
 * BdplDataFile.java
 * This is the data file implementation. We read from a file on disk
 * and store the contents of the file in memory in a byte array. We
 * maintain a pointer to the current position in the file. When data
 * is read, the file pointer is incremented. This class implements
 * the BdplFile interface, and thus is like the memfile, except that
 * the contents are read from disk. Since we know that files on disk
 * will have bytes, we do not need to maintain a pointer to the end
 * of the buffer.
 * @author akshay
 */
public class BdplDataFile implements BdplFile
{


    private ArrayList _data;
    private int _bit_pointer; // pointer to the next readable element

    /**
     * Creates a new instance of BdplDataFile
     */
    public BdplDataFile (String filename) throws BdplException
    {
        _data=new ArrayList();
        _bit_pointer=-1;
        try
        {

            FileInputStream file=new FileInputStream(filename);

            boolean flag=true;
            while(flag)
            {
                int size=file.available ();
                byte[] inp =new byte[size];
                int bytes_read=file.read (inp);
                for(int i=0;i<bytes_read;i++)
                {
                    _data.add (inp[i]);
                }
                if(bytes_read<=0)
                    flag=false;
                else
                {
                    _bit_pointer=0;
                }
            }
            file.close ();

        }
        catch(IOException e)
        {
            throw new BdplException (e.getMessage ());
        }

    }

    /** make a file froma bitsequence. limit to max_size bits */
    public BdplDataFile (String bitsequence,int max_size) throws BdplException
    {
        _data=new ArrayList();
        _bit_pointer=-1;

    }


    public BdplDataFile (BdplDataFile f,int maxsize) throws BdplException
```

```
    {
        _data=new ArrayList();
        _bit_pointer=-1;

    }

    public String read_n_bytes (int n) throws Exception
    {
        String ret="";
        if( (_bit_pointer+8*n) > 8*_data.size () )
            throw new Exception("dont have "+n+" bytes to read, have only " + ( 8*_data.size ()-
_bit_pointer)+" bits");
        while(n>0 && _bit_pointer<=8*(_data.size ()-1))
        {
            String s=read_n_bits(8);
            int c=0;
            for(int i=0;i<8;i++)
            {
                c=c<<1;
                if(s.charAt (i)=='1')
                    c+=1;
            }
            ret+=(char)c;
            n--;
        }
        return ret;
    }

    public String read_n_bits (int n) throws Exception
    {

        String ret="";

        while(_bit_pointer < 8*_data.size () && (_bit_pointer %8)!=0 && n > 0)
        {
            boolean b=get_nth_readable_bit (0);
            if(b)
                ret+='1';
            else
                ret+='0';

            _bit_pointer++;
            n--;
        }

        int start= n==0?0:(_bit_pointer)   / 8;
        int end  = n==0?0:(_bit_pointer+n) / 8;
        int rem  = n==0?0:(_bit_pointer+n) % 8;

        for(int i=start; i<_data.size () && i<end ;i++)
        {
            String s=Integer.toBinaryString ( Byte.parseByte (_data.get (i).toString ()));
            if(s.length ()>8)
                s=s.substring (s.length ()-8);

            int l=s.length ();
            for(int j=0;j<8-l;j++)
            {
                s='0'+s;
            }
            ret+=s;
            _bit_pointer+=8;
        }


        for(int i=0;i<rem;i++)
        {
            boolean b=get_nth_readable_bit (0);
            if(b)
                ret+='1';
            else
```

```
                ret+='0';

            _bit_pointer++;
        }

        return ret;
    }

    /** returns the next nth readable bit (0 indexed) */
    public boolean get_nth_readable_bit(int n) throws Exception
    {
        if( ((n + _bit_pointer) > 8*_data.size () ) || _bit_pointer < 0 )
            throw new Exception("index out of range");
        int off=(n+_bit_pointer)%8;
        byte mask= (byte)(1 << (7-off));
        return ( (Byte.parseByte ( _data.get ( (n+_bit_pointer)/8 ).toString () ) & mask)!=0);
    }

    public boolean eof()
    {
        return _bit_pointer>= 8*_data.size ();
    }

    public int num_readable_bits()
    {
        return (8*_data.size ())-_bit_pointer;
    }

}//end of class
```

```
import java.io.*;
import java.util.*;

/**
 * BdplDataFile.java
 * This is the data file implementation. We read from a file on disk
 * and store the contents of the file in memory in a byte array. We
 * maintain a pointer to the current position in the file. When data
 * is read, the file pointer is incremented. This class implements
 * the BdplFile interface, and thus is like the memfile, except that
 * the contents are read from disk. Since we know that files on disk
 * will have bytes, we do not need to maintain a pointer to the end
 * of the buffer.
 * @author akshay
 */
public class BdplDataFile implements BdplFile
{


    private ArrayList _data;
    private int _bit_pointer; // pointer to the next readable element

    /**
     * Creates a new instance of BdplDataFile
     */
    public BdplDataFile (String filename) throws BdplException
    {
        _data=new ArrayList();
        _bit_pointer=-1;
        try
        {

            FileInputStream file=new FileInputStream(filename);

            boolean flag=true;
            while(flag)
            {
                int size=file.available ();
                byte[] inp =new byte[size];
                int bytes_read=file.read (inp);
                for(int i=0;i<bytes_read;i++)
                {
                    _data.add (inp[i]);
                }
                if(bytes_read<=0)
                    flag=false;
                else
                {
                    _bit_pointer=0;
                }
            }
            file.close ();

        }
        catch(IOException e)
        {
            throw new BdplException (e.getMessage ());
        }

    }

    /** make a file froma bitsequence. limit to max_size bits */
    public BdplDataFile (String bitsequence,int max_size) throws BdplException
    {
        _data=new ArrayList();
        _bit_pointer=-1;

    }


    public BdplDataFile (BdplDataFile f,int maxsize) throws BdplException
```

```
    {
        _data=new ArrayList();
        _bit_pointer=-1;

    }

    public String read_n_bytes (int n) throws Exception
    {
        String ret="";
        if( (_bit_pointer+8*n) > 8*_data.size () )
            throw new Exception("dont have "+n+" bytes to read, have only " + ( 8*_data.size ()-
_bit_pointer)+" bits");
        while(n>0 && _bit_pointer<=8*(_data.size ()-1))
        {
            String s=read_n_bits(8);
            int c=0;
            for(int i=0;i<8;i++)
            {
                c=c<<1;
                if(s.charAt (i)=='1')
                    c+=1;
            }
            ret+=(char)c;
            n--;
        }
        return ret;
    }

    public String read_n_bits (int n) throws Exception
    {

        String ret="";

        while(_bit_pointer < 8*_data.size () && (_bit_pointer %8)!=0 && n > 0)
        {
            boolean b=get_nth_readable_bit (0);
            if(b)
                ret+='1';
            else
                ret+='0';

            _bit_pointer++;
            n--;
        }

        int start= n==0?0:(_bit_pointer)   / 8;
        int end  = n==0?0:(_bit_pointer+n) / 8;
        int rem  = n==0?0:(_bit_pointer+n) % 8;

        for(int i=start; i<_data.size () && i<end ;i++)
        {
            String s=Integer.toBinaryString ( Byte.parseByte (_data.get (i).toString ()));
            if(s.length ()>8)
                s=s.substring (s.length ()-8);

            int l=s.length ();
            for(int j=0;j<8-l;j++)
            {
                s='0'+s;
            }
            ret+=s;
            _bit_pointer+=8;
        }


        for(int i=0;i<rem;i++)
        {
            boolean b=get_nth_readable_bit (0);
            if(b)
                ret+='1';
            else
```

```
                ret+='0';

            _bit_pointer++;
        }

        return ret;
    }

    /** returns the next nth readable bit (0 indexed) */
    public boolean get_nth_readable_bit(int n) throws Exception
    {
        if( ((n + _bit_pointer) > 8*_data.size () ) || _bit_pointer < 0 )
            throw new Exception("index out of range");
        int off=(n+_bit_pointer)%8;
        byte mask= (byte)(1 << (7-off));
        return ( (Byte.parseByte ( _data.get ( (n+_bit_pointer)/8 ).toString () ) & mask)!=0);
    }

    public boolean eof()
    {
        return _bit_pointer>= 8*_data.size ();
    }

    public int num_readable_bits()
    {
        return (8*_data.size ())-_bit_pointer;
    }

}//end of class
```

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class BdplMain
{
    public static void usage ()
    {
        System.out.println ("================================================");
        System.out.println ("Bdpl Usage:");
        System.out.println ("java BdplMain [options] <file>");
        System.out.println ("\t -f : File specified as next argument");
        System.out.println ("\t -h : To display this help text");
        System.out.println ("================================================");
        System.exit (0);
    }
    public static void main (String[] args)
    {
        try
        {
            InputStream input = null;

            if(args.length > 0)
            {
                for(int argCount = 0; argCount < args.length; argCount++)
                {
                    if(args[argCount].equals ("-h"))
                    {
                        usage ();
                        System.exit (0);
                    }
                    else if(args[argCount].equals ("-f"))
                    {
                        if(args.length <= argCount+1)
                            usage();
                        input = new FileInputStream (args[argCount+1]);
                        argCount++;
                    }
                    else
                    {
                        usage ();
                        System.exit (0);
                    }
                }
            }
            else
            {
                usage ();
            }

            BdplLexer lexer = new BdplLexer (input);

            BdplParser parser = new BdplParser (lexer);
            parser.program ();


            CommonAST parseTree = (CommonAST)parser.getAST ();
            //System.out.println(parseTree.toStringList());
            //ASTFrame frame = new ASTFrame("AST for the BDPL Parser", parseTree);
            //frame.setVisible(false);


            BdplTreeParser treeParser = new BdplTreeParser ();
            treeParser.program (parseTree);
        }
        catch(Exception e)
        {
            System.err.println (e.getMessage ());
            return ;
            //sSystem.exit(0);
```

```
            //System.err.println("Exception: "+e);
        }

    }
}
```

```
import java.io.*;
import java.util.*;
/**
 * BdplMemFile.java
 * This is the mem file implementation. This class implements the
 * the BdplFile interface, and thus is like the datafile (and indeed
 * the two can e used interchangeably). This "file" does not correspond
 * to data on a disk, but must be initialized with a bitsequence in
 * memory. This file also has a way to build itself from BdplFile.
 * This type of file is used to represent bitsequences of objects
 * in memory.
 * @author akshay
 */
public class BdplMemFile implements BdplFile
{

    private String _data;
    private int _bit_pointer; // pointer to the next readable element
    private void init()
    {
        _data="";
    }

    /** Creates a new instance of BdplMemFile from a bitsequence*/
    public BdplMemFile (String bitsequence)
    {
        init();
        _data=bitsequence;
    }


    /** make a file of n bits from the given file without incrementing its pointer */
    public BdplMemFile (BdplFile f,int n) throws Exception
    {
        init();
        for(int i=0;i<n;i++)
        {
            if( f.get_nth_readable_bit (i))
            {
                _data+='1';
            }
            else
            {
                _data+='0';
            }

        }

    }

    public String read_n_bytes (int n) throws Exception
    {
        if( 8*n > _data.length ())
            throw new Exception("dont have "+n+" bytes to read, have only " + _data.length () + "
bits");
        String ret="";
        for(int i=0;i<n;i++)
        {
            ret += (char)Byte.parseByte (_data.substring (i*8,(i+1)*8),2);
        }
        _data=_data.substring (n*8);
        return ret;
    }

    public String read_n_bits (int n) throws Exception
    {
        if(n>_data.length ())
            throw new Exception("dont have "+n+" bits to read, have only " + _data.length () + "
bits");
        String ret="";
        if(n<_data.length ())
```

```
        {
            ret=_data.substring (0,n);
            _data=_data.substring (n);
        }
        else if(n==_data.length ())
        {
            ret=_data;
            _data="";
        }

        return ret;
    }

    /** returns the next nth readable bit (0 indexed) */
    public boolean get_nth_readable_bit(int n) throws Exception
    {
        if( n>_data.length ())
            throw new Exception("index out of range");

        return ( (_data.charAt (n)=='1') );
    }

    public boolean eof()
    {
        return (_data.length ()==0);
    }

    public int num_readable_bits()
    {
        return _data.length ();
    }

}//end of class
```

```
/*
 * Bitwise.java
 *
 */

import java.util.BitSet;
/**
 *
 * @author Bharadwaj Vellore
 */
public class Bitwise implements BdplLexerTokenTypes {

    private TypeChecker _typeChecker;
    /** Creates a new instance of Bitwise */
    public Bitwise(TypeChecker typeChecker) {
        _typeChecker = typeChecker;
    }

    public DataNodeAbstract eval(
            int operator,
            DataNodeAbstract operand1,
            DataNodeAbstract operand2) throws Exception
    {
        DataNodeAbstract result = null;
        int res = 0;

        if(!TypeChecker.isBasic(operand1.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }
        if(!TypeChecker.isBasic(operand2.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }

        Type resultType;

        switch(operator){
            case LLSH:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() << operand2.get_int_value();
                break;
            case LRSH:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() >>> operand2.get_int_value();
                break;
            case ALSH:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() << operand2.get_int_value();
                break;
            case ARSH:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() >> operand2.get_int_value();
                break;
            case ROR:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                //res = operand1.get_int_value() % operand2.get_int_value();
                break;
            case ROL:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand1.get_type_name());
                result = resultType.getDataNode();
                //res = operand1.get_int_value() % operand2.get_int_value();
```

```
                break;
            case BAND:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand2.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() & operand2.get_int_value();
                break;
            case BEOR:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand2.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() ^ operand2.get_int_value();
                break;
            case BIOR:
                resultType =
_typeChecker.getResultType(operand1.get_type_name(),operand2.get_type_name());
                result = resultType.getDataNode();
                res = operand1.get_int_value() | operand2.get_int_value();
                break;
            default:
                res = 0;
                //
                // This point will never be reached
                //
                assert(false);
        }

        if(result instanceof DataNodeInt){
            return new DataNodeInt(res);
        }else if(result instanceof DataNodeByte){
            return new DataNodeByte(res);
        }else if(result instanceof DataNodeBit){
            return new DataNodeBit(res);
        }else{
            //
            // This should never happen
            //
            assert(false);
            return null;
        }
    }

}
```

```
/**
 *
 * @author Akshay Pundle
 */

/**
 * Cache.java
 *
 * Simple class to cache bitsequences, array sizes etc
 */
public class Cache
{

    /** Creates a new instance of Cache with some data (that is the cached data) */
    public Cache (Object ob)
    {
        _valid=true;
        _object_data=ob;
    }
    /** Creates a new instance of Cache with NO cached data
     * if there's nothing in the cache, whatever is in it is invalid
     * so set valid=false here
     */
    public Cache()
    {
        _valid=false;
    }

    /** "set" the cache , i.e cache a new object */
    public void set(Object ob)
    {
        _valid=true;
        _object_data=ob;
    }

    /** retreive cached object as an integer */
    public int as_int()
    {
        return Integer.parseInt (_object_data.toString ());
    }

    /** retreive cached object as a string */
    public String as_string()
    {
        return _object_data.toString ();
    }

    /** retreive cached object as an object */
    public Object as_ob()
    {
        return _object_data;
    }

    /** set the cache as "valid"
     * have made this method private because
     * there shouldnt be any way of saying this from outside
     */
    private void valid()
    {
        _valid=true;
    }

    /** make the cache invalid */
    public void invalid()
    {
        _valid=false;
    }

    /** check if the cache is valid */
    public boolean is_valid()
    {
```

```
        return _valid;
    }

    /** class private data */
    /** is the object valid ? */
    private boolean _valid;
    /** the object data */
    private Object _object_data;
}
```

```
import antlr.collections.AST;
/**
 * DataNode.java
 *
 * This defines the interface which all data nodes should
 * adhere to (i.e this is the abstract base class for all
 * our concrete data node classes
 * @author Akshay Pundle
 */
public interface DataNode
{
    /** @return type name of this data node */
    public String get_type_name();
    /** @return the bits of this data node */
    public String get_bitsequence_value() ;

    /** @return size of this data in bits*/
    public int get_bit_size();

    /** @return size of this data as an integer (for expresion evaluation
     * purpose */
    public int get_int_value() throws Exception;

    /** @return the number of bits this type takes */
    public int get_fieldsize();

    /** set the expression for number of bits this type takes*/
    public void set_fieldsize(AST fieldsize);

    /** set valid ok nok */
    public void set_verify_then_else(AST verify_ast,AST then_ast , AST else_ast) throws Exception;

    /** sets the contaxt for evaluation of expressions for this data node*/
    public void set_context(VariableSymbolTable context);

    /** print a human readable form of this node */
    public String print();

    /** print with formatting
     * 0 = print() [debug formatted]
     * 1 = as string [useful to write to a file]
     * 2 = debug with strings [similar to 0, but with strings instad of bytes]
     * @return Formatted output ready to be printed
     */
    public String print(int format);
    public void set_name(String name);
    public String get_name();

    /** returns the maximum size this object can accept (hold) or -1 for infinity */
    public int get_max_accept() throws Exception;
    public void assign(DataNodeAbstract rhs) throws Exception;
    public void populate(BdplFile rhs) throws Exception;

}
```

```
/**
 *
 * @author Akshay Pundle
 */
import antlr.collections.*;
abstract public class DataNodeAbstract implements DataNode
{

    /** Creates a new instance  */
    private void init()
    {
        _fieldsize=0;
        _bitsize=0;
        _context=null;
        _fieldsize_ast=null;
        _name="";
        _verify_ast=null;
        _then_ast=null;
        _else_ast=null;
    }
    public DataNodeAbstract ()
    {
        init();
    }
    public DataNodeAbstract (DataNodeAbstract data)
    {
        init();
        _fieldsize=data._fieldsize;
        _bitsize=data._bitsize;
        _fieldsize_ast=data._fieldsize_ast;
        _then_ast=data._then_ast;
        _verify_ast=data._verify_ast;
        _else_ast=data._else_ast;
        _context=data._context;
        _name=data._name;

    }



    public Exception get_ex(String ex_string)
    {
        return new Exception(this.getClass().getName()  +": "+ex_string);
    }

    public void set_name(String name)
    {
        _name=name;
    }
    public String get_name()
    {
        return _name;
    }
    public void assign(DataNodeAbstract rhs) throws Exception
    {
        evaluate_fieldsize ();
        BdplMemFile rhs_file=new BdplMemFile(rhs.get_bitsequence_value ());
        populate(rhs_file);
        evaluate_verify_then_else ();
    }


    abstract public int get_max_accept() throws Exception;
    abstract public String get_type_name();
    abstract public String get_bitsequence_value();

    /** set the expression for number of bits this type takes*/
    public void set_fieldsize(AST fieldsize_ast)
    {
        _fieldsize_ast=fieldsize_ast;
    }
```

```
/** sets the contaxt for evaluation of expressions for this data node*/
public void set_context(VariableSymbolTable context)
{
    _context=context;
}

protected int evaluate_fieldsize () throws Exception
{
    BdplTreeParser par=new BdplTreeParser ();
    if(_context != null)
        par.varSymbTbl=_context;

    if(_fieldsize_ast != null)
    {

        int f=((DataNodeAbstract)par.expr (_fieldsize_ast)).get_int_value ();
        if( f>=0 && (_bitsize<=0 || f<_bitsize  ) )
        {
            _fieldsize=f;
        }
    }

    return _fieldsize;

}

public void set_verify_then_else(AST verify_ast,AST then_ast , AST else_ast)
{
    _verify_ast=verify_ast;
    _then_ast=then_ast;
    _else_ast=else_ast;
}

public void set_then(AST then_ast )
{
    _then_ast=then_ast;
}

public void set_else(AST else_ast )
{
    _else_ast=else_ast;
}

protected void evaluate_verify_then_else () throws Exception
{
    BdplTreeParser par=new BdplTreeParser ();
    if(_context != null)
        par.varSymbTbl=_context;

    if (_verify_ast != null && ((DataNodeAbstract)par.expr (_verify_ast)).get_int_value ()!=0)
    {
        if(_then_ast!= null) par.stmts (_then_ast);
    }
    else
    {
        if(_else_ast!= null) par.stmts (_else_ast);
    }

}



public int get_bit_size()  {return _bitsize;}

abstract public int get_int_value() throws Exception ;

/** @return the number of bits this type takes */
public int get_fieldsize() { return _fieldsize;}

/** set the number of bits this type takes
```

```
     * @return nonzero on error */
    public int set_fieldsize(int fieldsize) {_fieldsize=fieldsize;return 0;}

    abstract public void populate(BdplFile rhs) throws Exception ;

    /** print a human readable form of this node */
    abstract public String print();

    /** print a human readable form of this node */
    abstract public String print(int format);

    /** store the max size of the data in num of bits */
    protected int _fieldsize;
    /** stores the current size of the data in bits */
    protected int _bitsize;

    protected String _name;

    VariableSymbolTable _context;

    AST _fieldsize_ast;
    AST _verify_ast;
    AST _then_ast;
    AST _else_ast;

}
```

```
/*
 * DataNodeArray.java
 *
 */

/**
 *
 * @author Akshay Pundle
 */

import java.util.*;
import antlr.collections.AST;

/** Implementsa an array node */
public class DataNodeArray extends DataNodeAbstract
{
    /** common initializers */
    private void init()
    {
        _bseq_cache = new Cache();
        _bsize_cache=new Cache();
        _intvalue_cache=new Cache();
        _children= new ArrayList() ;
        _delayed_size=false;
        _size_evaled=false;

    }

    private DataNodeArray ()
    {
        super();
        init();
    }


    /** Creates a new instance of DataNodeArray
     * the dummy variable is only to keep a refernce of
     * what type the elements of this array are
     */
    public DataNodeArray (DataNodeAbstract dummy)
    {
        super();
        init();
        _dummy=dummy;
    }

    /** Creates a new instance of DataNodeArray
     * just like this array node
     */
    public DataNodeArray (DataNodeArray node)
    {
        super(node);
        init();
        _dummy=node._dummy;
        _fieldsize_ast=node._fieldsize_ast;
        _verify_ast=node._verify_ast;
        _scope_var_pointer=node._scope_var_pointer;
    }

    public DataNodeArray (DataNodeAbstract dummy,AST _arr_size_expr)
    {
        super();
        init();
        _arr_size_expr_ast=_arr_size_expr;
        _isunlimited=false;
        _delayed_size=true;
        _size_evaled=false;
        _dummy=dummy;
    }
    public DataNodeArray (DataNodeAbstract dummy,int arr_size) throws Exception
    {
```

```
    super();
    init();
    _dummy=dummy;
    add_n_elements(arr_size);
}

private void add_n_elements (int n) throws Exception
{
    for(int i=0;i< n; i++ )
    {
        DataNodeAbstract child=null;
        if(_dummy.getClass ().getCanonicalName ().equals ("DataNodeBit"))
        {
            child=new DataNodeBit (_dummy);
        }
        else if(_dummy.getClass ().getCanonicalName ().equals ("DataNodeByte"))
        {
            child=new DataNodeByte (_dummy);
        }
        else if(_dummy.getClass ().getCanonicalName ().equals ("DataNodeInt"))
        {
            child=new DataNodeInt (_dummy);
        }
        else if(_dummy.getClass ().getCanonicalName ().equals ("DataNodeArray"))
        {
            throw new BdplException ("array or arrays not allowed");
        }
        else if(_dummy.getClass ().getCanonicalName ().equals ("DataNodeStruct"))
        {
            child=new DataNodeStruct ( (DataNodeStruct) _dummy  );
            ((DataNodeStruct) child).get_scope_var ().set_parent (_scope_var_pointer);

        }
        child.set_context (_scope_var_pointer);
        _children.add (child);


    }

}

/** sets a pointer to the new symbol table for this struct */
public void set_scope(VariableSymbolTable spv)
{
    _scope_var_pointer=spv;
}
/** get the symbol table pointer for this struct */
public VariableSymbolTable get_scope_var()
{
    return _scope_var_pointer;
}

public DataNodeAbstract get_dummy()
{
    return _dummy;
}
/*
/** should invoke copy constructor for base class *
public DataNodeArray (DataNodeArray data)
{
    super(data);
    init();
    _children = data._children;
}*/


public int get_max_accept() throws Exception
{
    if(_isunlimited)
        return -1;
    else
```

```
    {
        int max=0;
        for(int i=0;i<_children.size ();i++)
        {
            int ret=((DataNodeAbstract)_children.get (i)).get_max_accept ();
            if(ret<0)
                return ret;
            else
                max+=ret;

        }
        return max;
    }


}
/* lets see if the base class function works
public void assign(DataNodeAbstract rhs)
{
    if(rhs.getClass ().getCanonicalName ().equals (("DataNodeArray")))
    {
        // copying from array to array
    }
    evaluate_verify_then_else();
}*/

/** return the unique typename */
public String get_type_name() {return "ARRAY";}

public String get_bitsequence_value()
{
    if(_bseq_cache.is_valid ())
    {
        return _bseq_cache.as_string ();
    }
    String ret="";
    for(int i=0;i<_children.size() ; i++ )
    {
        ret=ret+ ((DataNodeAbstract)_children.get (i)).get_bitsequence_value ();
    }
    _bseq_cache.set (ret);
    return ret;
}
public int get_bit_size()
{
    if(_bsize_cache.is_valid ())
    {
        return _bseq_cache.as_int();
    }
    int ret=0;
    for(int i=0;i<_children.size() ; i++ )
    {
        ret+=((DataNodeAbstract)_children.get (i)).get_bit_size ();
    }
    _bsize_cache.set (ret);
    return ret;
}


public int get_int_value() throws Exception
{
    if(_intvalue_cache.is_valid ())
    {
        return _intvalue_cache.as_int ();
    }

    int ret=0;
    String t=get_bitsequence_value();
    if(t.length ()<_offset)
    {
        throw get_ex("Underflow in get_int_val. offset too big ?");
```

```
    }
    else
    {
        int i=_offset;
        for(;i<t.length () && i<32+_offset ; i++)
        {
            ret=ret<<1;
            if(t.charAt (i)=='1')
            {
                ret+=1;
            }
        }
        /** may be one after the last element in the array. Beware ! */
        _offset=i;
    }

    _intvalue_cache.set (ret);
    return ret;
}

/** returns true if this array is a * array */
public boolean get_is_unlimited() {return _isunlimited;}
public void set_is_unlimited(boolean isunlimited) {_isunlimited=isunlimited;}

/** returns size of the array in num of elements */
public int get_size () throws Exception
{
    if(_delayed_size && !_size_evaled)
    {
        _size_evaled=true;
        evaluate_and_resize ();

    }
    return _children.size ();
}

/** returns the ith element from the array */
public DataNodeAbstract get_element(int i) throws Exception
{
    if(i>=_children.size ())
    {
        throw get_ex("index out of bounds");
    }
    return (DataNodeAbstract)_children.get (i);
}

/** sets the ith element of the array
 * you cannot set the size of the star-array by just accessing
 * an unexisting element. Thus, we will thro an exception here
 * regardless of whether we have anormal or a star array.
 * @throws exception
 */
public void set_element(int i,DataNodeAbstract data) throws Exception
{
    if(data.get_type_name () != _dummy.get_type_name ())
    {
        throw get_ex("incompatible types ! all elemens of array must be of the defined type.");
    }

    if(_delayed_size && !_size_evaled)
    {
        _size_evaled=true;
        evaluate_and_resize ();

    }
    if(i>=_children.size ())
    {
        throw get_ex("Index Out Of Bound in set_element");
    }
    _children.set(i,data);
    _invalid_cache();
```

```
        evaluate_verify_then_else();
    }

    /** appends an element to the end of the array */
    public void append_element(DataNodeAbstract data) throws Exception
    {
        if(_delayed_size && !_size_evaled)
        {
            _size_evaled=true;
            evaluate_and_resize ();

        }
        if(data.get_type_name () != _dummy.get_type_name ())
        {
            throw get_ex("incompatible types ! all elemens of array must be of the defined type.");
        }

        _children.add(data);
        _invalid_cache();
        evaluate_verify_then_else();
    }

    /** returns the type of (all) the elements of the array
     * string for now. the return value sould be a "type" class */
    public String get_element_type_name() { return _dummy.get_type_name () ; }

    /** returns offset of the "current pointer' witin the array.*/
    public int get_offset() {return _offset; }

    /** sets curret pointer within the array */
    public void set_offset(int offset) {_offset=offset;_intvalue_cache.invalid ();}

    /** private funciton to invalidate all caches */
    private void _invalid_cache()
    {
        _bseq_cache.invalid ();
        _bsize_cache.invalid ();
        _intvalue_cache.invalid ();
    }
    public String print()
    {
        String ret=get_type_name ()+" of " + _dummy.get_type_name ()+"\n[\n";
        for(int i=0;i<_children.size() ; i++ )
        {
            ret+= "\t"+String.valueOf (i)+" => " + ((DataNodeAbstract)_children.get
(i)).print().replaceAll ("\n","\n\t")+",\n";
        }
        return ret+"]\n";
    }


    /** print formatted */
    public String print(int format)
    {
        if(format==0)
        {
            return print ();
        }
        else if(format ==1 )
        {
            return Utils.bits_as_string (get_bitsequence_value ());
        }
        else
        {
            return "";
        }
    }

    public void populate (BdplFile rhs) throws Exception
    {
        BdplFile temprhs=rhs;
```

```
    boolean read_only_fieldsize=false;
    int num_reading=0;

    /** if our fieldsize if less than what the children want,
     * we will
     *  1) make a copy of the data limited to ourfieldsize bits
     *  2) get the children to populate on this copy
     *  3) in the end detect how much input the children have consumed
     *      an consume an equal amount form the actual source
     */
    evaluate_fieldsize ();
    int ma=get_max_accept ();
    if( _fieldsize>0  && (_fieldsize < get_max_accept ()  || _isunlimited || ma==-1) )
    {
        BdplMemFile mem_file=new BdplMemFile(rhs,_fieldsize);
        read_only_fieldsize=true;
        temprhs=mem_file;
        num_reading=_fieldsize;
    }
    int i=0;
    if(_isunlimited)
    {
        int num_readable=temprhs.num_readable_bits();
        while( temprhs.num_readable_bits ()>0  )
        {
            add_n_elements(1);
            ((DataNodeAbstract)_children.get (_children.size ()-1)).populate (temprhs);
            if(temprhs.num_readable_bits () == num_readable)
                break; // we arnt reading any data, better get outta here
            else
                num_readable=temprhs.num_readable_bits ();
        }
    }
    else
    {
        evaluate_and_resize();
        // its ok if we even read 0 bytes in every iteration, no chance
        // of an infinite loop here
        while( temprhs.num_readable_bits ()>0 && i<_children.size () )
        {
            ((DataNodeAbstract)_children.get (i)).populate (temprhs);
            i++;
        }
    }

    if(read_only_fieldsize)
    {
        rhs.read_n_bits (num_reading-temprhs.num_readable_bits ()); // consume bits from rhs
    }
    evaluate_verify_then_else();


}

private void evaluate_and_resize () throws Exception
{
    if(_delayed_size)
    {
        BdplTreeParser par=new BdplTreeParser ();
        if(_scope_var_pointer != null)
            par.varSymbTbl=_scope_var_pointer;

        int siz=((DataNodeAbstract)par.expr (_arr_size_expr_ast)).get_int_value ();
        if(siz>_children.size ())
        {
            add_n_elements (siz-_children.size ());
        }

    }
}
```

```
    public AST get_ast()
    {
        return _arr_size_expr_ast;
    }

    /**  class private data */
    private AST _arr_size_expr_ast;
    boolean _delayed_size;
    boolean _size_evaled;
    private VariableSymbolTable _scope_var_pointer;

   /** current pointer within the array (in bits) */
    private int _offset;

    /** elements of the array */
    private List _children;

    /** true if this array is a resizable (growable) array */
    private boolean _isunlimited;

    /** stores a dummy variable of the type of each element of the array */
    private DataNodeAbstract _dummy;

    /** bit sequence  cache */
    private Cache _bseq_cache;

    /** bit size cache */
    private Cache _bsize_cache;

    /** cache calculated int value */
    private Cache _intvalue_cache;

}
```

```
/*
 * DataNodeBit.java
 *
 */

/**
 *
 * @author Akshay Pundle
 */
public class DataNodeBit extends DataNodeAbstract
{

    void init()
    {
        _bitsize=1;
        _fieldsize=1;
    }
    /** list all constructors here */
    /** Creates a new instance of DataNodeBit */
    public DataNodeBit () {super();init();_data=0;}
    public DataNodeBit (DataNodeAbstract d) {super(d);init();_data=0;}

    public int get_max_accept()
    {
        return 1;
    }


    public void assign(DataNodeAbstract rhs) throws Exception
    {
        String b=rhs.get_bitsequence_value ();
        int a=0;
        evaluate_fieldsize ();
        int start=0,end=_fieldsize-1;

        if(rhs.getClass ().getCanonicalName (). equals ("DataNodeBit")  ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeByte") ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeInt"))
        {
            start=(b.length ()>=_fieldsize)?b.length ()-_fieldsize:0;
            end=b.length ()-1;
        }
        for(int i=start;i<=end && i<b.length ();i++)
        {
            a=a<<1;
            if(b.charAt (i)=='1')
                a+=1;

        }
        _data=a;
        evaluate_verify_then_else();

    }

    /** construct from int */
    public DataNodeBit (int data) {super(); init();_data=data;}

    /** copy constructor */
    public DataNodeBit (DataNodeBit data)
{super(data);init();_data=data._data;_fieldsize=data._fieldsize;_bitsize=data._bitsize;}


    /** return the unique typename */
    public String get_type_name() {return "TYPE_BIT";}

    public String get_bitsequence_value()
    {
        String s=Integer.toBinaryString (_data);
        return ""+s.charAt (0);
    }
    public int get_int_value() {return _data;}
```

```
    public String print()
    {
        return get_bitsequence_value ();
    }
    /** print formatted */
    public String print(int format)
    {
        if(format==0)
        {
            return print ();
        }
        else if(format ==1 )
        {
            return Utils.bits_as_string (get_bitsequence_value ());
        }
        else
        {
            return "";
        }
    }


    public void populate (BdplFile rhs) throws Exception
    {
        evaluate_fieldsize ();
        if(_fieldsize <=0)
        {
            evaluate_verify_then_else ();
            return;
        }

        if(rhs.num_readable_bits ()>=1)
        {
            String next_bit=rhs.read_n_bits (1);
            if(next_bit.charAt (0)=='1')
                _data=1;
            else
                _data=0;
        }
        evaluate_verify_then_else();
    }

    private int _data;

}
```

```
/*
 * DataNodeByte.java
 *
 */

/**
 *
 * @author Akshay Pundle
 */
public class DataNodeByte extends DataNodeAbstract
{
    void init ()
    {
        _bitsize=8;
        _fieldsize=8;
    }
    /** Creates a new instance of DataNodeByte */
    public DataNodeByte ()
    {
        super ();
        init ();
        _data=0;
    }

    /** construct from int */
    public DataNodeByte (int data)
    {
        super ();
        init ();
        _data=data&0xff;
    }

    /** copy constructor */
    public DataNodeByte (DataNodeAbstract data)
    {
        super (data);
        init ();
        _bitsize=data._bitsize;
        _fieldsize=data._fieldsize;
    }

    public int get_max_accept() throws Exception
    {
        evaluate_fieldsize ();
        return _fieldsize;
    }

    public void assign(DataNodeAbstract rhs) throws Exception
    {
        evaluate_fieldsize ();
        String b=rhs.get_bitsequence_value ();
        int a=0;
        int start=0,end=_fieldsize-1;

        if(rhs.getClass ().getCanonicalName (). equals ("DataNodeBit")  ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeByte") ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeInt"))
        {
            start=(b.length ()>=_fieldsize)?b.length ()-_fieldsize:0;
            end=b.length ()-1;
        }
        for(int i=start;i<=end && i<b.length ();i++)
        {
            a=a<<1;
            if(b.charAt (i)=='1')
                a+=1;

        }
        _data=a;
        evaluate_verify_then_else();
    }
```

```
    /** return the unique typename */
    public String get_type_name () {return "TYPE_BYTE";}

    public String get_bitsequence_value ()
    {
        String s=Integer.toBinaryString (_data);
        if((s.length () >= _bitsize))
        {
            return s.substring ( s.length ()-_bitsize, s.length ());
        }
        else
        {
            int l=s.length ();
            for(int i=0;i<_bitsize-l;i++)
            {
                s='0'+s;
            }
            return s;
        }
    }
    public int get_int_value () {return _data;}

    public String print()
    {
        return String.format ("0x%x",_data);
    }


    /** print formatted */
    public String print(int format)
    {
        if(format==0)
        {
            return print ();
        }
        else if(format ==1 )
        {
            //return get_bitsequence_value ()+"\n";
            return Utils.bits_as_string (get_bitsequence_value ());
        }
        else
        {
            return "";
        }
    }

    public void populate (BdplFile rhs) throws Exception
    {
        evaluate_fieldsize ();
        if(rhs.num_readable_bits ()>=_fieldsize && _fieldsize!=0)
        {
            String bits=rhs.read_n_bits (_fieldsize);
            _data=(int) Integer.parseInt (bits,2);
        }
        evaluate_verify_then_else();
    }

    private int _data;

}
```

```
/*
 * DataNodeInt.java
 *
 */

/**
 *
 * @author Akshay Pundle
 */
public class DataNodeInt extends DataNodeAbstract
{
    void init()
    {
        _bitsize=32;
        _fieldsize=32;
    }

    /** Creates a new instance of DataNodeInt */
    public DataNodeInt () {super();init();_data=0;}

    /** construct from int */
    public DataNodeInt (int data) {super();init();_data=data; }

    /** copy constructor */
    public DataNodeInt (DataNodeAbstract data) {super(data);init();_bitsize=data._bitsize;
_fieldsize=data._fieldsize;}

    public int get_max_accept() throws Exception
    {
        evaluate_fieldsize ();
        return _fieldsize;
    }

    public void assign(DataNodeAbstract rhs) throws Exception
    {
        evaluate_fieldsize ();
        String b=rhs.get_bitsequence_value ();
        int a=0;
        int start=0,end=_fieldsize-1;

        if(rhs.getClass ().getCanonicalName (). equals ("DataNodeBit")  ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeByte") ||
            rhs.getClass ().getCanonicalName ().equals ("DataNodeInt"))
        {
            start=(b.length ()>=_fieldsize)?b.length ()-_fieldsize:0;
            end=b.length ()-1;
        }
        for(int i=start;i<=end && i<b.length ();i++)
        {
            a=a<<1;
            if(b.charAt (i)=='1')
                a+=1;

        }
        _data=a;
        evaluate_verify_then_else();
    }

    /** return the unique typename */
    public String get_type_name() {return "TYPE_INT";}
    public String get_bitsequence_value()
    {
        String s=Integer.toBinaryString (_data);
        if((s.length () >= _bitsize))
        {
            return s.substring ( s.length ()-_bitsize, s.length ());
        }
        else
        {
            int l=s.length ();
            for(int i=0;i<_bitsize-l;i++)
```

```
        {
            s='0'+s;
        }
        return s;
    }
}
public int get_int_value() {return _data;}

public String print()
{
    return String.valueOf (_data);
}

/** print formatted */
public String print(int format)
{
    if(format==0)
    {
        return print ();
    }
    else if(format ==1 )
    {
        return Utils.bits_as_string (get_bitsequence_value ());
    }
    else
    {
        return "";
    }
}

public void populate (BdplFile rhs) throws Exception
{
    evaluate_fieldsize ();
    if(_fieldsize <=0)
    {
        evaluate_verify_then_else();
        return;
    }

    if(rhs.num_readable_bits ()>=_fieldsize)
    {
        String bits=rhs.read_n_bits (_fieldsize);
        if(Globals.little_endian)
        {
            _data=0;
            for(int i=0;i<4;i++)
            {
                _data=_data<<8;
                if(bits.length ()>=8)
                {
                    String s=bits.substring (bits.length ()-8);
                    _data+=Integer.parseInt (s,2);
                    if(bits.length ()>8)
                        bits=bits.substring (0,bits.length ()-8);
                    else
                    {
                        evaluate_verify_then_else ();
                        return ;
                    }


                }
                else
                {
                    _data+=Integer.parseInt (bits);
                }
            }
        }
        else
        {
            _data=(int)Long.parseLong (bits,2);
```

```
            }
            evaluate_verify_then_else();
        }
    }

    private int _data;

}
```

```
/*
 * DataNodeStruct.java
 *
 */

/**
 *
 * @author Akshay Pundle
 */

import java.util.*;
public class DataNodeStruct extends DataNodeAbstract
{


    /** common initializers */
    private void init()
    {
        _bseq_cache = new Cache();
        _bsize_cache=new Cache();
        _intvalue_cache=new Cache();
        _children= new HashMap() ;
        _sequence=new ArrayList();
        _scope_type_pointer=null;
        _scope_var_pointer=null;
        _type_name="TYPE_STRUCT";

    }

    /** sets a pointer to the new symbol table for this struct */
    public void set_scope(VariableSymbolTable spv,TypeSymbolTable spt)
    {
        _scope_type_pointer=spt;
        _scope_var_pointer=spv;
    }
    /** get the symbol table pointer for this struct */
    public VariableSymbolTable get_scope_var()
    {
        return _scope_var_pointer;
    }
    /** get the symbol table pointer for this struct */
    public TypeSymbolTable get_scope_type()
    {
        return _scope_type_pointer;
    }
    /** Creates a new instance of DataNodeStruct */
    public DataNodeStruct ()
    {
        super();
        init();
    }


    /** should invoke copy constructor for base class */
    public DataNodeStruct(DataNodeStruct data) throws Exception
    {
        super(data);
        init();
        _scope_var_pointer=new VariableSymbolTable();
        _scope_type_pointer=data._scope_type_pointer;
        for(int i=0;i<data._sequence.size ();i++)
        {
            DataNodeAbstract child=null;
            _sequence.add (data._sequence.get (i));
            DataNodeAbstract d = (DataNodeAbstract)data._children.get (data._sequence.get (i));

            if(d.getClass ().getCanonicalName ().equals ("DataNodeBit"))
            {
                child=new DataNodeBit(d);
            }
            else if(d.getClass ().getCanonicalName ().equals ("DataNodeByte"))
```

```
            {
                child=new DataNodeByte(d);
            }
            else if(d.getClass ().getCanonicalName ().equals ("DataNodeInt"))
            {
                child=new DataNodeInt(d);
            }
            else if(d.getClass ().getCanonicalName ().equals ("DataNodeArray"))
            {
                if(((DataNodeArray)d).get_ast()!=null)
                {
                    child=new DataNodeArray( ((DataNodeArray)d).get_dummy(),
((DataNodeArray)d).get_ast()  );
                    child._else_ast=d._else_ast;
                    child._context=d._context;
                    child._fieldsize_ast=d._fieldsize_ast;
                    child._then_ast=d._then_ast;
                    child._verify_ast=d._verify_ast;
                    child._fieldsize=d._fieldsize;

                }
                else
                {
                    child=new DataNodeArray( ((DataNodeArray)d).get_dummy(),
((DataNodeArray)d).get_size()  );
                    child._else_ast=d._else_ast;
                    child._context=d._context;
                    child._fieldsize_ast=d._fieldsize_ast;
                    child._then_ast=d._then_ast;
                    child._verify_ast=d._verify_ast;
                    child._fieldsize=d._fieldsize;
                }
                ((DataNodeArray)child).set_scope (_scope_var_pointer);
                ((DataNodeArray)child).set_is_unlimited ( ((DataNodeArray)d).get_is_unlimited ());
            }
            else if(d.getClass ().getCanonicalName ().equals ("DataNodeStruct"))
            {
                child=new DataNodeStruct( (DataNodeStruct) d  );
                ((DataNodeStruct)child).get_scope_var ().set_parent (_scope_var_pointer);
            }
            _children.put (data._sequence.get (i),child );
            child.set_context (_scope_var_pointer);
            child.set_name (d.get_name ());
            _scope_var_pointer.insert ((String)data._sequence.get (i),child);

        }
        _type_name=data._type_name;

    }

    public int get_max_accept () throws Exception
    {

        int max=0;
        for(int i=0;i<_sequence.size();i++)
        {
            int ret=((DataNodeAbstract) _children.get ( _sequence.get (i))).get_max_accept ();
            if(ret<0)
                return ret;
            else
                max+=ret;

        }
        return max;
    }
    /* lets see if the base class function works
    public void assign(DataNodeAbstract rhs)
    {
        evaluate_verify_then_else();
    }
     */
```

```
    /** return the unique typename */
    public String get_type_name()
    {
        return _type_name;
    }
    /** set the type name of this struct */
    public void set_type_name(String type_name)
    {
        _type_name=type_name;
    }

    /** @return concatenation of bit_sequence of all children */
    public String get_bitsequence_value()
    {
        if(_bseq_cache.is_valid ())
        {
            return _bseq_cache.as_string ();
        }
        String ret="";
        for (int i=0;i<_sequence.size ();i++)
        {
            ret+= ((DataNodeAbstract)_children.get ( (String)_sequence.get
(i) )).get_bitsequence_value();
        }
        _bseq_cache.set (ret);
        return ret;
    }

    /** @return current size of struct in bits */
    public int get_bit_size ()
    {
        if(_bsize_cache.is_valid ())
        {
            return _bsize_cache.as_int ();
        }

        int ret=0;

        for (int i=0;i<_sequence.size ();i++)
        {
            ret+= ((DataNodeAbstract)_children.get ( _sequence.get (i))).get_bit_size ();
        }
        _bsize_cache.set (ret);
        return ret;
    }

    /** gets the next 4 bytes (as int) from the struct. increments
     * ofset pointer.
     * @return the next (upto) 4 bytes from the struct as int */

    public int get_int_value () throws Exception
    {
        if(_intvalue_cache.is_valid ())
        {
            return _intvalue_cache.as_int ();
        }

        int ret=0;
        String t=get_bitsequence_value ();
        if(t.length ()<_offset)
        {
            throw get_ex ("Underflow in get_int_val. offset too big ?");
        }
        else
        {
            int i=_offset;
            for(;i<t.length () && i<32+_offset ; i++)
            {
                ret=ret<<1;
                if(t.charAt (i)=='1')
                {
```

```
                ret+=1;
            }
        }
        /** may be one after the last element in the array. Beware ! */
        _offset=i;
    }
    _intvalue_cache.set (ret);
    return ret;
}

/** size of struct presumably in number of children*/
public int get_size()
{
    return _children.size ();
}

/** returns child(element of the struct) given name of the child */
public DataNodeAbstract get_child_by_name (String name) throws Exception
{
    if(_children.containsKey (name))
    {
        return (DataNodeAbstract) _children.get (name);

    }
    else
    {
        throw get_ex ("no child found named "+name);
    }
}

/** initialize / set a child by name */
public void set_child_by_name(String name,DataNodeAbstract  value) throws Exception
{
    if(_children.containsKey (name))
    {
        throw get_ex("child already declared : "+name);
    }
    else
    {
        _children.put (name,value);
        _sequence.add (name); //remember the sequence number of this child
    }
    evaluate_verify_then_else();
}

/** returns child(element of the struct) given number of the child */
public DataNodeAbstract get_child_by_num(int num) throws Exception
{
    if(num>_children.size ())
        throw new Exception("index out of bounds in get_child_by_num");
    return (DataNodeAbstract)_children.get(_sequence.get (num));
}

/** @return when implemented, will return all children(elements of the struct) */
public DataNodeAbstract get_all_children(int num) throws Exception
{
    throw get_ex("get_all_children not implemented");
}

/** returns number of elements the struct has */
public int get_num_children()
{
    return _children.size ();
}

/** returns the type of a child of the struct
 * string for now. the return value sould be a "type" class */
public String get_child_type_by_name(String name) throws Exception
{
    if(_children.containsKey (name))
    {
```

```
            return ((DataNodeAbstract) _children.get (name)).get_type_name ();

        }
        else
        {
            throw get_ex("no child found named "+name);
        }
    }

    /** current pointer within the struct */
    public int get_offset() {return _offset; }

    /** set current pointer within the struct */
    public void set_offset(int offset) {_offset=offset;}

    public String print()
    {
        String ret=get_type_name ()+"\n{\n";
        for (int i=0;i<_sequence.size ();i++)
        {
            String key= (String)_sequence.get (i);
            ret+= "\t"+  key+" => " + ((DataNodeAbstract)_children.get (key)).print().replaceAll
("\n","\n\t") +",\n";
        }
        return ret+"}\n";
    }

    /** print formatted */
    public String print(int format)
    {
        if(format==0)
        {
            return print ();
        }
        else if(format ==1 )
        {
            return Utils.bits_as_string (get_bitsequence_value ());
        }
        else
        {
            return "";
        }
    }

    public void populate (BdplFile rhs) throws Exception
    {

        BdplFile temprhs=rhs;
        boolean read_only_fieldsize=false;
        int num_reading=0;

        /** if our fieldsize if less than what the children want,
         * we will :
         *  1) make a copy of the data limited to ourfieldsize bits
         *  2) get the children to populate on this copy
         *  3) in the end detect how much input the children have consumed
         *      an consume an equal amount form the actual source
         */
        int ma=get_max_accept();
        evaluate_fieldsize ();
        if( _fieldsize>0 && ( _fieldsize < ma || ma==-1))
        {
            BdplMemFile mem_file=new BdplMemFile(rhs,_fieldsize);
            read_only_fieldsize=true;
            temprhs=mem_file;
            num_reading=_fieldsize;
        }

        int i=0;
        while( temprhs.num_readable_bits ()>0 && i<_sequence.size ())
        {
```

```
            DataNodeAbstract child= (DataNodeAbstract)_children.get (_sequence.get (i) );
            child.populate (temprhs);
            i++;
        }

        if(read_only_fieldsize)
        {
            rhs.read_n_bits (num_reading-temprhs.num_readable_bits ()); // consume bits from rhs
        }
        evaluate_verify_then_else();
    }

    private int _offset;

    private String _type_name;

    private HashMap _children;

    /** stores the sequence of children */
    private List _sequence;

    /** bit sequence  cache */
    private Cache _bseq_cache;

    /** bit size cache */
    private Cache _bsize_cache;

    /** cache calculated int value */
    private Cache _intvalue_cache;

    private VariableSymbolTable _scope_var_pointer;
    private TypeSymbolTable _scope_type_pointer;

}
```

```
/*
 * Globals.java
 *
 * Created on December 15, 2007, 10:16 AM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

/**
 *
 * @author akshay
 */
public class Globals
{
    public Globals()
    {
        little_endian=true;
    }
    /** is everything supposed to be done in little endian ? */
    public static boolean little_endian=true;
}
```

```
/*
 * Logical.java
 *
 */

/**
 *
 * @author Bharadwaj Vellore
 */
public class Logical implements BdplLexerTokenTypes{

    private TypeChecker _typeChecker;
    /** Creates a new instance of Logical */
    public Logical(TypeChecker typeChecker) {
        _typeChecker = typeChecker;
    }

    public boolean shortckt(int operator, DataNodeAbstract operand) throws Exception{
        if(!TypeChecker.isBasic(operand.get_type_name())){
            throw new BdplException(operand.get_name()+" is not of a basic type");
        }

        if(LAND == operator && operand.get_int_value()==0) return true;
        if(LOR == operator && operand.get_int_value()!=0) return true;

        return false;
    }

    public DataNodeAbstract eval(
            int operator,
            DataNodeAbstract operand1,
            DataNodeAbstract operand2) throws Exception
    {
        boolean res;

        if(!TypeChecker.isBasic(operand1.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }
        if(!TypeChecker.isBasic(operand2.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }

        switch(operator){
            case LAND:
            res = (operand1.get_int_value()==0?false:true) && (operand2.get_int_value()==0?
false:true);
                break;
            case LOR:
                res = (operand1.get_int_value()==0?false:true) || (operand2.get_int_value()==0?
false:true);
                break;
            default:
                res = false;
                //
                // This point will never be reached
                //
                assert(false);
        }

        return new DataNodeBit(res==true?1:0);
    }
}
```

```
/*
 * Relational.java
 *
 */

/**
 *
 * @author Bharadwaj Vellore
 */
public class Relational implements BdplLexerTokenTypes{

    private TypeChecker _typeChecker;
    /** Creates a new instance of Relational */
    public Relational(TypeChecker typeChecker) {
        _typeChecker = typeChecker;
    }

    public DataNodeAbstract eval(
            int operator,
            DataNodeAbstract operand1,
            DataNodeAbstract operand2) throws Exception
    {
        boolean res;

        if(!TypeChecker.isBasic(operand1.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }
        if(!TypeChecker.isBasic(operand2.get_type_name())){
            throw new BdplException(operand1.get_name()+" is not of a basic type");
        }

        switch(operator){
            case GT:
                res = operand1.get_int_value() > operand2.get_int_value();
                break;
            case LT:
                res = operand1.get_int_value() < operand2.get_int_value();
                break;
            case GTE:
                res = operand1.get_int_value() >= operand2.get_int_value();
                break;
            case LTE:
                res = operand1.get_int_value() <= operand2.get_int_value();
                break;
            case EQUALITY:
                res = operand1.get_int_value() == operand2.get_int_value();
                break;
            case INEQUALITY:
                res = operand1.get_int_value() != operand2.get_int_value();
                break;
            default:
                res = false;
                //
                // This point will never be reached
                //
                assert(false);
        }

        return new DataNodeBit(res==true?1:0);
    }

}
```

```
/*
 * Type.java
 *
 */

/**
 *
 * @author Preethi Narayan, Akshay Pundle
 */
import antlr.collections.*;
public class Type
{
    private String _type;
    private DataNodeAbstract _array_dummy;
    private DataNodeAbstract _struct_node;
    private AST _struct_ast;

    /** Creates a new instance of Type */
    public Type(String type)
    {
        if( type=="bit" ||
            type=="byte" ||
            type=="int"
            )
        {
            _type=type;
        }
        else
        {
            _type="";
        }
    }

     /**
     *  constructor which takes the subtree and the type name
     *  type must be struct here
     */
    public Type(String type,DataNodeAbstract node)
    {
        if(type.startsWith ("ARRAY"))
        {
            _type=type;
            _array_dummy=node;
            _struct_node=null;
        }
        else if(type.startsWith ("struct"))
        {

            //_type=type;
            //_struct_node=node;
            //_array_dummy=null;
            _struct_node=null;
            _array_dummy=null;
            _type=null;
        }
        else
        {
            _struct_node=null;
            _array_dummy=null;
            _type=null;
        }


    }


    public Type(String type,AST struct_ast)
    {
        if(type.startsWith ("struct"))
        {
            _struct_node=null;
```

```
        _array_dummy=null;
        _type=type;
        _struct_ast=struct_ast;
    }
    else
    {
        _struct_node=null;
        _array_dummy=null;
        _type=null;
    }


}

/**
 *method to get the type of the type object
 */
public DataNodeAbstract getDataNode() throws Exception
{
    /*if(!st.contains (_type))
        throw new Exception(_type+" type not defined");*/

    if(_type.equals("int"))
    {
        return new DataNodeInt();
    }
    else if(_type.equals("bit"))
    {
        return new DataNodeBit();
    }
    else if(_type.equals("byte"))
    {
        return new DataNodeByte();
    }
    else if(_type.startsWith("ARRAY"))
    {
        throw new Exception("getDataNode() not implemented for struct : use getDataNode(int)");
    }
    else if(_type.startsWith("struct"))
    {
        throw new Exception("get data node not implemented for struct : use get_ast");
    }
    throw new Exception("bad type!");
}



public DataNodeAbstract getDataNode(int arr_size) throws Exception
{
    if(_type.startsWith("ARRAY"))
    {
        return new DataNodeArray(_array_dummy,arr_size);
    }
    else
    {
        throw new Exception("getDataNode(int) is only for making arrays");
    }
}


public AST get_ast()
{
    return _struct_ast;
}
/**
 *method to return the type as a string
 */
public String getTypeName()
{
    return _type;
}
```

```
}
```

```
/*
 * TypeChecker.java
 *
 */

/**
 *
 * @author Bharadwaj Vellore
 */
public class TypeChecker {

    private TypeSymbolTable _typeSymbTbl;
    private TypeConverter _typeConvert;

    /** Creates a new instance of TypeChecker */
    public TypeChecker(TypeSymbolTable typeSymbTbl, TypeConverter typeConvert) {
        _typeSymbTbl = typeSymbTbl;
        _typeConvert = typeConvert;
    }

    public static boolean isBasic(String type){
        return (
                (type == "TYPE_INT")?
                    true:
                    (
                        (type == "TYPE_BYTE")?
                            true:
                            (
                                (type == "TYPE_BIT")?
                                    true:
                                    false
                            )
                    )
                );
    }

    public Type getResultType(String type1, String type2) throws Exception
    {
        return _typeSymbTbl.get(_typeConvert.get(type1,type2));
    }

}
```

```
/*
 * TypeConverter.java
 *
 */

/**
 *
 * @author Bharadwaj Vellore
 */

import java.util.*;

public class TypeConverter {

    private Map _map;
    private Map _intMap;
    private Map _byteMap;
    private Map _bitMap;

    /** Creates a new instance of TypeConverter */
    public TypeConverter() {
        _intMap = new HashMap();
        _intMap.put("TYPE_BYTE","int");
        _intMap.put("TYPE_BIT","int");
        _intMap.put("TYPE_INT","int");

        _bitMap = new HashMap();
        _bitMap.put("TYPE_BYTE","byte");
        _bitMap.put("TYPE_BIT","bit");
        _bitMap.put("TYPE_INT","int");

        _byteMap = new HashMap();
        _byteMap.put("TYPE_BYTE","byte");
        _byteMap.put("TYPE_BIT","byte");
        _byteMap.put("TYPE_INT","int");

        _map = new HashMap();
        _map.put("TYPE_INT",_intMap);
        _map.put("TYPE_BYTE",_byteMap);
        _map.put("TYPE_BIT",_bitMap);
    }

    public String get(String ltype, String rtype){
        HashMap map = (HashMap)_map.get(ltype);
        return (String)map.get(rtype);
    }
}
```

```
/*
 * TypeSymbolTable.java
 *
 */

/**
 *
 * @author Preethi Narayan, Akshay Pundle
 */

import java.util.*;
import antlr.collections.*;
public class TypeSymbolTable
{
    /**
     * Creates a new instance of TypeSymbolTable
     */
    private Map _the_table;
    private TypeSymbolTable _parent; // pointer to parent
    public TypeSymbolTable()
    {
        _the_table=new HashMap();
        _the_table.put("int",new Type("int")); // predefined types
        _the_table.put("bit",new Type("bit"));
        _the_table.put("byte",new Type("byte"));
        _the_table.put("ARRAY:byte",new Type("ARRAY",new DataNodeByte()));
        _the_table.put("ARRAY:int",new Type("ARRAY",new DataNodeInt()));
        _the_table.put("ARRAY:bit",new Type("ARRAY",new DataNodeBit()));
        _parent=null;

    }

    public void set_parent(TypeSymbolTable parent)
    {
        _parent=parent;
    }
    public TypeSymbolTable get_parent()
    {
        return _parent;
    }
    /**
     * check if entry exists in the table otherwise make an entry into this table and the variable
symbol table
     */
    public void insert(String type_name,Type type_ob) throws Exception
    {
        if(_the_table.containsKey (type_name) )
        {
            throw new Exception(type_name+" already exists in symbol table");
        }
        else
        {
             _the_table.put (type_name,type_ob);
        }

    }
    /**
     *method to retieve a type node from the type symbol table
     */
    public Type get(String type) throws Exception
    {
        if(_the_table.containsKey(type))
        {
            return (Type) _the_table.get (type);
        }
        else
        {
            if(_parent!=null)
            {
                return _parent.get (type);
```

```
            }
            else
            {
                throw new Exception(type+" : undefined type");
            }
        }
    }

    /** check if the symbol table containg this type already */
    private boolean contains(String type)
    {
        if(_the_table.containsKey(type))
        {
            return true;
        }
        else
        {
            if(_parent!=null)
            {
                return _parent.contains(type);

            }
            else
            {
                return false;
            }
        }
    }
}
```

```
/*
 * TypeSymbolTable.java
 *
 */

/**
 *
 * @author Preethi Narayan, Akshay Pundle
 */

import java.util.*;
import antlr.collections.*;
public class VariableSymbolTable
{
    /**
     * Creates a new instance of TypeSymbolTable
     */
    private Map _the_table;
    private VariableSymbolTable _parent;

    public VariableSymbolTable()
    {
        _the_table=new HashMap();
        _parent=null;
    }

    public void set_parent(VariableSymbolTable parent)
    {
        _parent=parent;
    }
    public VariableSymbolTable get_parent()
    {
        return _parent;
    }


    /**
     * check if entry exists in the table otherwise make an entry into this table and the variable
symbol table
     */
    public void insert(String id,DataNodeAbstract datanode) throws Exception
    {
        if(_the_table.containsKey (id) )
        {
            throw new Exception(id+" already exists in symbol table");
        }
        else
        {
             _the_table.put (id,datanode);
        }

    }
    /**
     *method to retieve a type node from the type symbol table
     */
    public DataNodeAbstract get(String id) throws Exception
    {
        if(_the_table.containsKey(id))
        {
            return (DataNodeAbstract) _the_table.get (id);
        }
        else
        {
            if(_parent!=null)
            {
                return _parent.get (id);

            }
            else
            {
                throw new Exception(id+" : undefined identifier");
```

```
            }
        }
    }

    /** check if the symbol table containg this type already */
    public boolean contains(String id)
    {
        return _the_table.containsKey (id);
    }
}
```

8.2 Sample Programs and Test cases

8.2.1 An ID3 Tag Parser

```
int tagsize;
int framesize;

struct MP3_FILE
{
        struct MP3_ID3
        {
                byte[3] id;
                byte[2] version;
                byte flags;
                byte[4] size satisfies {1<5} then {
                        tagsize = size[0];
                        tagsize = ((tagsize << 7) | (size[1] & 0x7F));
                        tagsize = ((tagsize << 7) | (size[2] & 0x7F));
                        tagsize = ((tagsize << 7) | (size[3] & 0x7F));
                };
        }mp3_id3;

        struct FRAME
        {
                byte[4] frameheader_id satisfies {frameheader_id[0] == 0x54} then {
                        framesize = 0;
                }else{          };
                byte[4] size satisfies {1} then {
                        framesize = size[0];
                        framesize = ((framesize << 8) | (size[1] & 0xFF));
                        framesize = ((framesize << 8) | (size[2] & 0xFF));
                        framesize = ((framesize << 8) | (size[3] & 0xFF));
                };
                byte[2] flags;
        struct TEXT_FRAME
            {
                  byte encoding;
                  byte[framesize-2] string satisfies {frameheader_id[0] == 0x54} then {
                              if(frameheader_id[1] == 0x49) {
                                     print("Song Title: ");printstring(string);print("::");
                              }
                              if(frameheader_id[1] == 0x41) {
                                     print("Album: ");printstring(string);print("::");
                              }
                              if(frameheader_id[1] == 0x43 && frameheader_id[2] == 0x4F &&
 frameheader_id[3] == 0x4E) {
                                     print("Genre: ");printstring(string);print("::");
                              }
                              if(frameheader_id[1] == 0x50 && frameheader_id[2] == 0x45 &&
 frameheader_id[3] == 0x32) {
                                     print("Performer: ");printstring(string);print("::");
                              }
                              if(frameheader_id[1] == 0x43 && frameheader_id[2] == 0x4F &&
 frameheader_id[3] == 0x4D) {
                                     print("Composer: ");printstring(string);print("::");
                              }
```

```
                    };
                    byte empty;
            }text;
    }[*]textFrame;
        byte[*] data;
}mp3_file;

file "x:/test/ChaleChalo.mp3" inp_file;
read(inp_file,mp3_file);
```

8.2.2 Objdump

```
int symtab_offset;
int symtab_size;

int strtab_offset;
int strtab_size;
int i;
int j;
int k;
bit flag;
byte[128] name;

struct symbol_table_entry
{
        int st_name;
        int st_value;
        int st_size;
        byte st_info;
        byte st_other;
        byte[2] st_shndx;
}Elf32_Sym;

struct elf_file
{

        struct elf_header
        {
                byte[16]        e_ident;
                byte[2]         e_type;
                byte[2]         e_machine;
                int                     e_version;
                byte[8]         e_other_stuff1;
                int                     e_shoff;
                int                     e_flags;
                byte[2]         e_ehsize;
                byte[2]         e_phentsize;
                byte[2]         e_phnum;
                byte[2]         e_shentsize;
                byte[2]         e_shnum;
                byte[2]         e_shstrndx;
        }elf_hd;

        byte[elf_hd.e_shoff-52] e_other_stuff2;

        struct section_header
        {
                int sh_name;
                int sh_type;
                int sh_flags;
                int sh_addr;
                int sh_offset satisfies{sh_type==2 || sh_type==3 }
                        then
                        {
                                if(sh_type==2)
                                        symtab_offset=sh_offset;
                                else
                                        strtab_offset=sh_offset;
                        };
                int sh_size satisfies{sh_type==2 || sh_type==3 }
```

```
                        then
                        {
                                if(sh_type==2)
                                        symtab_size=sh_size;
                                else
                                        strtab_size=sh_size;
                        };


                int sh_link;
                int sh_info;
                int sh_addralign;
                int sh_entsize;
        } [elf_hd.e_shnum[0]] se_array;

        byte[ symtab_offset - ( ( $#se_array*40)+elf_hd.e_shoff) ] e_other_stuff3;

}elf_file;

struct elf_file_1
{
        byte[symtab_offset] e_other_stuff4;
        type struct symbol_table_entry[symtab_size/16] symbol_table;
        byte[strtab_offset-(symtab_offset + symtab_size)] e_other_stuff5;
        byte[strtab_size] string_table;
}elf_file_1;

file "x:/test/tut.o" inp_file;
read(inp_file,elf_file);
file "x:/test/tut.o" inp_file;
read(inp_file,elf_file_1);

for(i=0;i<$#elf_file_1.symbol_table;i=i+1)
{
        if(elf_file_1.symbol_table[i].st_name != 0 )
        {
                flag=1;
                for(k=elf_file_1.symbol_table[i].st_name;flag==1;k=k+1)
                {
                        for(j=0;j<128;j=j+1)
                        {
                                //name[j]=0;
                        }
                        if(elf_file_1.string_table[k] == 0 )
                        {
                                flag=0;
                                continue;
                        }
                        else
                        {
                                printstring(elf_file_1.string_table[k]);
                        }
                }

                print(" ");
        }
        print(elf_file_1.symbol_table[i]);
}
```

8.2.3 MPEG Transport Stream File Parser

```
//
// This is an MPEG Transport Stream Processor
//
int count;
int video; int videopackets;
int audio; int audiopackets; int auxpackets;
video = 8191; audio = 17;
count = 0;
struct TransportPacket{
```

```
        byte sync satisfies {sync == 0x47} then {count=count+1;}
                else { count=count+1; print("Sync Error in packet:");print(count);print("::");};
        bit[3] auxBits;
        int pid (fieldsize 13) satisfies {pid == video || pid == audio}
                then {
                                if(pid == video) {
                                        videopackets = videopackets + 1;
                                } else {
                                        audiopackets = audiopackets + 1;
                                }
                        } else {
                                auxpackets = auxpackets + 1;
                        };
        bit[2] adaptationFieldControl;
        bit[2] scramblingStatus satisfies {1} then {if(scramblingStatus[1] != 0){print("Packet
");print(count);print(" is scrambled::");}};
        int continuityCounter (fieldsize 4)
                satisfies {(count > 1 && continuityCounter == ((tp[count-2].continuityCounter + 1) &
0xF))} then {}
                else {print("Discontinuity in packet: "); print(count);print("::");};
        byte[184] payload;
}[*] tp;

file "x:\test\mpeg_test.ts" inputFile ;
read(inputFile,tp);
print($#tp); print(" packets found::"); print(videopackets); print(" are video, ");
print(audiopackets); print(" are audio, ");
print(auxpackets); print(" are auxillary::");
```

8.2.4 Unlimited array test program

```
struct outer
{
        byte[*] first;
        byte[*] second;
}b;

file "x:/test/file_test.dat" inp_file;

int i;
for(i=0;i<10;i=i+1)
{
        read(inp_file,b.second);
        printstring(b);
}
```

8.2.5 Loop Test

```
int i;
int j;
i=0;
for(;;){
        for(j=0;j<5;j=j+1){
                if(j<i){
                        print(j);
                }else{
                        break;
                }
        }
        i=i+1;
        if(i<10) {
                print(i);
                continue ;
        }else{
                break;
        }
}
```

8.2.6 Simple Test for Satisfies Clause

```
int a (fieldsize 5) satisfies { a==5 && a!=6 && a==7} then { print("ok");} else {print("not ok");};

file "x:/test/file_test.dat" inp_file;
read(inp_file,a);
```

8.2.7 Struct Test

```
struct Outer
{
        int b_1;
        struct Inner
        {
                byte i_2_1;
                byte i_2_2;
        } inner_ob_2;
        int c_3;
} ob1;

struct Outer1
{
        int b_1;
        struct Outer
        {
                byte i_2_1;
                byte i_2_2;
        } inner_ob_2;
        int c_3;
        struct Inner
        {
                int h;
        }na;
        type struct Outer[4] n_outer;
        type struct Inner new_inner;
}[5] ob2;
int[*]  aa;

type struct Outer[5] arr_outer;
```

8.2.8 Relational Operator Test

```
print(42 > 21);
print(42 < 21);
print(42 >= 21);
print(42 <= 21);
print(42 == 21);
print(42 != 21);
```

8.2.9 Arithmetic Operator Test

```
print(3 + 7 + 5 + 0); //15
print(2 * 3 * 4 * 5); // 120
print(2+ 3 * 4+ 5); // 19
print(2* 3 + 4 * 5); // 26

print(3 - 7 + 8 -9); // -5
print(3 -7- 8 -9); //-21
print(3/2 + 4*7 - 12); // 17
```

8.2.10 Simple fieldize test

```
byte[*] a (fieldsize 5+3);
```

```
file "x:/test/file_test.dat" inp_file;
read(inp_file,a);
print(a);
```