Objective

The objective of this project is to construct a tool that allows for the creation and editing of bit fields.

Project Description

Binary data can be difficult to work with without manually writing out the bit positions and what their values should be. This smalltalk based tool will allow for the editing of a bit field’s individual bits. From there the tool will be able to convert the field of bits into other forms such as ascii, hex and decimal if preferred. It will also feature bitwise logical expressions that can be evaluated at binary level. Two fields can be operated on and produce an altered field based on the logical operation applied to it. This field can overwrite one of the previous fields or be used for referential purposes without changing anything.

The tool also features an annotation system that allows the labeling and organization of bit fields as objective entities. Individual annotated fields could be rearranged and reordered without the need to rewrite each of the bits by hand. The annotations will be displayed as a list within the tool that will highlight when clicked on to show the position and size of the bitfield.

The heart of the project’s creation interface is an interpreter that allows for the creation of annotated bitfield templates. These templates can be created from various grammar specs as well as custom ones defined by the user. The interpreter will take in a specification and produce a set of annotated bit fields. Consequently these specifications could also be saved and loaded from a file. By doing this it will allow for the templating and exporting of common and custom structures. Examples of this feature are templating the TCP packet structure, or identifying and labeling a master boot record or executable file.

Grammar Specification

Bitfield {

 BitStructure

 = Label "{" Declaration\* "};"

 Declaration

 = ArrayDeclaration

 | FieldDeclaration

 | BitStructure

CompoundDeclaration

 = Label "->" Declaration ";"

ArrayDeclaration

 = Label "(" BinaryArray ");"

 | Label "(" HexArray ");"

FieldDeclaration

 = Label ":" number ";"

 | Label ":" FieldFill ";"

FieldFill

 = number "[" Byte "]"

 | number "[" "0".."1" "]"

BinaryArray (Binary array)

 = ("0".."1" ",")\* "0".."1"

HexArray (Hex Array)

 = (Byte ",")\* Byte

Byte

 = hexDigit hexDigit

label

 = letter\*

number

 = digit+

String

 = "\""alnum\*"\""

}

Point Allocation

|  |  |
| --- | --- |
| Bit array |  |
| * Converts to hex
 | 5 |
| * Converts to binary
 | 5 |
| * Converts to Ascii
 | 5 |
| * Can combine with others in a managed set to reform the original array in desired format
 | 5 |
| Annotations |  |
| * Changed Annotations update in the editor windows properly
 | 5 |
| * Annotations show up for not only single fields but a collection of fields as well
 | 5 |
| Editor GUI system |  |
| * Field editor shows structures as annotated subdivisions of the whole set
 | 10 |
| * Fields can be individually displayed as either binary or hex
 | 10 |
| * Code window for the interpreted grammar with functioning buttons to build structures
 | 10 |
| * Bitwise operations with proper bit field updates
 | 10 |
| Grammar specification (PEG) | 25 |
| * Is the PEG easy to read?
 | 10 |
| * Can clean code be written with it?
 | 10 |
| * Is it simple to use?
 | 10 |
| Interpreter |  |
| * Properly Built AST
 | 5 |
| * Output an annotated bit field from a spec
 | 5 |
| * Output an annotated bit field with pre-set array data
 | 5 |
| * Output an entire structure with multiple annotated arrays in one structure
 | 10 |
| * Save and load the bit structure from a file
 | 5 |
| Points Total | 160 |

Grading Scale

|  |  |
| --- | --- |
| A | 90-100% |
| B | 80-89% |
| C | 70-79% |
| D | 60-69% |
| F | 59% or lower |

Sample of Grammar in use

TCP {

 SourcePortNumber : 16[0];

 DestinationPortNumber : 16[0];

 SequenceNumber : 16[0];

 AcknowledgementNumber :16[0];

 WindowSize : 4[0];

 Reserved : 6[0];

 SingleBits {

 URG : 1;

 ACK : 1;

 PSH : 1;

 RST : 1;

 SYN : 1;

 FIN : 1;

 };

 WindowSize : 4[0];

 TCPChecksum : 16[0];

 UrgentPointer : 16[0];

 OptionsVariableLength : 32;

 Data (0A,5F,3C,66,45,82,73,5A,CC,EF,FD,EA);

};