Part 2 - Coding Concepts Questions - Maneesh Wijewardhana

2.1

- void createAndAddRectangle(xmlNode *node, List *list)
- Filename: SVGHelpers.c, Line: 62-99, Repo: CIS2750W22_Project, Branch: A1-A3

```
void createAndAddRectangle(xmlNode *node, List* list) {
  Rectangle* r = (Rectangle*)malloc(sizeof(Rectangle));
  // from LinkedListAPI.c
  r->otherAttributes = initializeList(&attributeToString,
&deleteAttribute, &compareAttributes);

  //for strtok
  char* units = NULL;

  for (xmlAttr* attr = node->properties; attr != NULL; attr = attr-
>next) {
    if (strcmp((char*)attr->name, "x") == 0) {
      //if first elm has units, we assume units for all and vice versa
      r->x = strtof((char*)attr->children->content, &units);
    } else if (strcmp((char*)attr->name, "y") == 0) {
      r->y = strtof((char*)attr->children->content, NULL);
    } else if (strcmp((char*)attr->name, "width") == 0) {
      r->width = strtof((char*)attr->children->content, NULL);
    } else if (strcmp((char*)attr->name, "height") == 0) {
      r->height = strtof((char*)attr->children->content, NULL);
    } else { //if none, we know its other attributes
      // from LinkedListAPI.c
      insertBack(r->otherAttributes, createAttribute(attr));
    }
  }
  //now if that first units for r->x was not null, we can set those
units for it
  if (units != NULL) {
    strcpy(r->units, units);
  }
  //add it now to rectangle list
  insertBack(list, r);
}
```

- The purpose of this function was add all attributes and units of a rectangle into the rectangle list of the SVG being read
    - It acts as more of a standalone function than a helper function as it declutters createSVG and makes it more modular
    - It allocates memory for a Rectangle struct and then loops through the properties of a node looking for the elements “x”, “y”, “width”, “height” as well as otherAttributes
    - To do this, I used strtof() as I also needed to check for units and strtof() allowed me to check for units on the first “x” element and parse it, at the end if units was NULL, we know not to add units but if it did catch units, we strcpy() the units over
    - Finally, it inserts the Rectangle struct into the list of rectangles in the actual SVG image
- To test this function, I utilized SvgToString() using an SVG image that contains rectangles in it
    - Steps for testing started off with first adding this function to my createSVG() function and calling it when looping through the xmlnodes until “rect” was found
    - To go in depth with testing, I chose an SVG image with rectangles in it, then in my mainTester.c I called createSVG() using that file, then I used SVGToString() on the returned SVG struct and was able to see if the attributes/elements were correctly parsed
    - I used multiple files as test cases such as rect.svg which only contains one rectangle without any units. This was a good first test case as it allowed me to make sure my parsing for each attribute/element of the rectangle was successful. Another test case was rect_with_units.svg which imitated rect.svg but had a units field for the rectangle. This allowed me to see if my strtof() properly parsed the ending of the x, y, width, and height values and added it to the final list. The file rects.svg was another one that I used as a test case as it allowed me to see if multiple rectangles in an SVG was able to be parsed using the loop in createSVG()
    - I would record the test results using SvgToString() and copying the results of that for each test case into a text file
    - To use these results, I would compare the actual xml code of the SVG image being tested and match up each attribute/element of the real image to the string that was recorded for that file
        - If all elements and attributes had the same values and did not miss anything, I knew that the function was done and fully tested

- bool setAttributeWrapper(char *filename, char *elemType, int elementIndex, char *newOtherAttr, char *newOtherAttrVal);
- Filename: SVGHelpers.c, Line: 692-722, Repo: CIS2750W22_Project, Branch: A3

```
bool setAttributeWrapper(char *filename, char *elemType, int
elementIndex, char *newOtherAttr, char *newOtherAttrVal) {
  bool ret = false;
  SVG *img = createValidSVG(filename, "parser/svg.xsd");
  elementType type;
  // check what type we need to add
  if (strcmp("Rectangles", elemType) == 0) {
    type = RECT;
  } else if (strcmp("Circles", elemType) == 0) {
    type = CIRC;
  } else if (strcmp("Paths", elemType) == 0) {
    type = PATH;
  } else if (strcmp("Groups", elemType) == 0) {
    type = GROUP;
  } else {
```

```
    type = SVG_IMG;
  }
  // make new attr and copy over value
  Attribute *newAttr = (Attribute *)calloc(1, sizeof(Attribute));
  newAttr->name = (char *)calloc(strlen(newOtherAttr) + 1 +
strlen(newAttr->value), sizeof(char));
  strcpy(newAttr->name, newOtherAttr);
  strcpy(newAttr->value, newOtherAttrVal);
  // set the attribute
  setAttribute(img, type, elementIndex, newAttr);
  // now validate to make sure before writing
  if (validateSVG(img, "parser/svg.xsd")) {
    ret = true;
    writeSVG(img, filename);
  }
  deleteSVG(img);
  return ret;
}
```

- The purpose of this function was to be able to add or edit an attribute from an existing SVG file on the node.js server
  - The function acts as a wrapper function for the required function setAttribute()
  - It first creates a valid SVG struct using the file on the server which gets passed in. It then compares a string passed in that represents the element type that needs to be updated which can be Rectangles, Circles, Paths, Groups, or the SVG image itself (this is what setAttribute() needs). Then the function allocates memory for a new attribute and copies over the passed in strings for the attribute name and value. It then calls the main setAttribute() function using the SVG image allocated, the element type that is being edited, the index of the element, and the new attribute struct that was allocated and modified. Lastly, it validates the final SVG, writes the image to the same file, frees the memory and returns a bool if the process was successful or not.
  - This return value is needed because if the validation fails, then the entire function will return false and will not write the new SVG image to the file
  - The passed in values such as filename, element type, element index, new attribute, and new attribute value come from the various drop down lists and forms in the HTML and JavaScript code on the client. Those values then get sent to the node.js server where the C function gets called
- To test this function, I used client side JavaScript and also dummy data in my mainTester.c to send hardcoded values for the new attribute name and value
  - I first tested using mainTester.c where I passed in rect.svg, “Rectangles” as a string, 0 as the index, “fill” as a string, and “not-none” as a string. I then called setAttributeWrapper() and checked to see if rect.svg had a different fill value that changed from “none” to not-non”. After this, I was able to use files such as rects.svg for changing indices, quad01.svg for other shapes, and also tried setting new attributes for shapes that did not have any in the first place (all using mainTester.c)
  - On the JavaScript side, I also tested to see if the function translated properly
  - To do this I created an endpoint on the node.js server that when hit, calls the C function using the request query parameters which were hardcoded on the client side.
  - For example, when sending a GET request to /editAttr, I used rect.svg, 0, fill, not-none as hardcoded values on the client side

- I then used console.log() on the data that was sent back to the client side and made sure the values were all correct as well as checking the actual SVG file that was saved to make sure the same values or new ones were added
- To record these testing results, I would first document the original file such as rect.svg and pay close attention to the other attributes of the rectangle shape. I would then hardcode a new attribute value or create a new one and call the function with the right parameters such as rect.svg, “Rectangles”, 0, “fill”, “not-none”, and record the newly created file
- After recording the results, I would be able to use them by comparing the two recorded files side by side and making sure that the only property that changed was the other attribute of the rectangle shape where “fill” “none” changed to “fill” “not-none”

2.2 - validateSVG()

- Filename: SVGParser.c, Line: 750-768, Repo: CIS2750W22_Project, Branch: A2
- Helpers Filename: SVGHelpers.c, Line: 190-446, Repo: CIS2750W22_Projects, Branch: A2
- **ANNOTATIONS ARE IN THE CODE SNIPPETS BOLDED**

```
bool validateSVG(const SVG *img, const char *schemaFile) {
 // check for null image and schema first
  FOLLOWING HEADER CONSTRAINT WHER FILENAME AND SCHEMA CANNOT BE NULL
  if (img == NULL || schemaFile == NULL) {
    return false;
  }
  // header constraints check using validate helper functions
  ALL CONSTRAINTS SPECIFIED IN SVGParser.h CHECKED HERE (EXPLAINED IN
DETAIL BELOW)
  IT DOES NOT MATTER THAT WE VALIDATE THE SPEC FIRST OR THE XML DOC AS
IF WE VALIDATE XML FIRST, IT WILL NOT CHECK FOR NEGATIVE CIRCLE RADIUS
FOR EXAMPLE BUT OUR VALIDATION HELPER WILL CATCH THAT.
  if (!isValidAttr(img->otherAttributes) || !isValidRect(img-
>rectangles) || !isValidCircle(img->circles) || !isValidPath(img-
>paths) || !isValidGroup(img->groups)) {
    return false;
  }
  CONVERTING IMAGE TO XML IN ORDER TO VALIDATE AGAINST LIBXML
(EXPLAINED IN DETAIL BELOW)
  // now check xsd
  xmlDoc *doc = svgToXmlDoc(img);
  NOW VALIDATING CREATED XML AGAINST SCHEMA FILE (EXPLAINED IN DETAIL
BELOW)
  int ret = isValidXml(doc, schemaFile);
  BASED ON IF MY HELPER RETURNS 0 OR ANYTHING ELSE, IT DETERMINES IF
FILE IS XML IS VALID OR NOT (EXPLAINED IN DETAIL BELOW)
  xmlFreeDoc(doc);
  if (ret == 0) {
    return true;
  } else {
    return false;
  }
  THIS IMPLEMENTATION IS CORRECT BECAUSE WE PROPERLY CHECK EACH SHAPE
AND ATTRIBUTE AND IF ANY OF THEM ARE FALSE, WE ALSO RETURN FALSE.
```

OTHERWISE, WE ARE ABLE TO CONVERT THE IMAGE TO XML AND VALIDATE AGAINST THE SCHEMA IN WHICH WE CAN EVALUATE ITS RETURN VALUE TO DETERMINE IF THE SVG IS VALID OR NOT.
}

HELPER FUNCTIONS USED FOR validateSVG()

```
bool isValidAttr(List *list) {
  // check for null list first
  MAKING SURE THE OTHER ATTRIBUTES LIST IS NOT NULL AS SPECIFIED BY SVGParser.h
  if (list == NULL) {
    return false;
  }
  // now iterate thru and check for null values
  Attribute *attr = NULL;
  ListIterator iter = createIterator(list);
  ITERATING THROUGH ALL ATTRIBUTES IN LIST UNTIL WE REACH THE END AND MAKING SURE THE NAME PROPERTY IS NOT NULL SPECIFIED BY SVGParser.h.
while ((attr = nextElement(&iter)) != NULL) {
    if (attr->name == NULL) {
      return false;
    }
  }
  return true;
  THIS IMPLEMENTATION IS CORRECT AS THIS WAY OF ITERATING MAKES SURE I REACH THE END AND ALSO FOR OTHER ATTRIBUTES, ONLY NAME CANNOT BE NULL WHEREAS VALUE CAN BE EMPTY
}

bool isValidRect(List *list) {
  // check for null list first
  MAKING SURE THE RECTANGLE LIST IS NOT NULL AS SPECIFIED BY SVGParser.h
  if (list == NULL) {
    return false;
  }
  // now iterate thru and check for null values
  Rectangle *r = NULL;
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  ITERATING THROUGH ALL RECTANGLES IN LIST UNTIL WE REACH THE END AND MAKING SURE THE WIDTH AND HEIGHT ARE NOT LESS THAN 0 AS SPECIFIED BY SVGParser.h. ALSO CHECKING IF ITS OTHER ATTRIBUTES LIST IS NULL AND USING HELPER FUNCTION ABOVE TO CHECK IF ITS OTHER ATTRIBUTES HAS A NULL NAME AS SPECIFIED BY SVGParser.h
  while ((r = nextElement(&iter)) != NULL) {
    if (r->width < 0 || r->height < 0 || r->otherAttributes == NULL || !isValidAttr(r->otherAttributes)) {
```

```
      return false;
    }
  }
  return true;
  THIS IMPLEMENTATION IS CORRECT BECAUSE I FOLLOWED THE SVG SPEC
PRECISELY AND AFTER TESTING WITH NEGATIVE WIDTH, HEIGHT, NULL OTHER
ATTR, AND NULL OTHER ATTRIBUTE NAME, THE FUNCTION RETURNED FALSE
}

bool isValidCircle(List *list) {
  // check for null list first
  MAKING SURE CIRCLE LIST ITSELF IS NOT NULL AS SPECIFIED BY
SVGParser.h
  if (list == NULL) {
    return false;
  }
  // now iterate thru and check for null values as well as negative
radius
  Circle *c = NULL;
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  ITERATING THROUGH ALL CIRCLES IN IMAGE UNTIL NULL AND CHECKING IF
ITS RADIUS IS LESS THAN O AS SPECIFIED BY SVGParser.h. ALSO MAKING
SURE ITS OTHER ATTRIBUTES LIST IS NOT NULL AS WELL AS USING MY OTHER
ATTRIBUTE HELPER FUNCTION TO CHECK IF ITS ATTRIBUTE NAME IS NOT NULL
AS SPECIFIED BY SVGParser.h
  while ((c = nextElement(&iter)) != NULL) {
    if (c->r < 0 || c->otherAttributes == NULL || !isValidAttr(c-
>otherAttributes)) {
      return false;
    }
  }
  return true;
  THIS IMPLEMENTATION IS CORRECT AS I FOLLOWED THE SVG CONSTRAINTS
PRECISELY AND TESTING WITH NEGATIVE RADIUS, NULL OTHER ATTRIBUTES, AND
NULL OTHER ATTRIBUTE NAMES, THE FUNCTION RETURNED FALSE
}

bool isValidPath(List *list) {
  // check for null list first
  CHECK IF PATH LIST IS NULL AS SPECIFIED BY SVGParser.h
  if (list == NULL) {
    return false;
  }
  // now iterate thru and check for null values
  Path *p = NULL;
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
```

```
  ITERATING THROUGH ALL PATHS IN IMAGE UNTIL NULL AND CHECKING IF ITS
DATA IS NULL OR ITS OTHER ATTRIBUTES IS NULL OR IF ITS OTHER ATTRIBUTE
NAME IS NULL AS SPECIFIED BY SVGParser.h
  while ((p = nextElement(&iter)) != NULL) {
    if (p->data == NULL || p->otherAttributes == NULL || !
isValidAttr(p->otherAttributes)) {
      return false;
    }
  }
  return true;
  THIS IMPLEMENTATION IS CORRECT AS TESTING A PATH WITH THE INVALID
CONDITIONS, THE FUNCTION RETURNED FALSE BUT TRUE OTHERWISE
}

bool isValidGroup(List *list) {
  // check for null list first
  CHECK FOR NULL GROUP LIST FIRST AS SPECIFIED BY SVGParser.h
  if (list == NULL) {
    return false;
  }
  // now iterate thru and check for null values using above helper
funcs
  Group *g = NULL;
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  ITERATE THROUGH ALL GROUPS IN IMAGE UNTIL NULL AND UTILIZE THE ABOVE
HELPER FUNCTIONS TO CHECK IF THERE OTHER ATTRIBUTES RETURNED FALSE,
RECTANGLES, CIRCLES, PATHS, AND GROUPS WITHIN RETURNED FALSE.
  while ((g = nextElement(&iter)) != NULL) {
    if (!isValidAttr(g->otherAttributes) || !isValidRect(g-
>rectangles) || !isValidCircle(g->circles) || !isValidPath(g->paths)
|| !isValidGroup(g->groups)) {
      return false;
    }
  }
  return true;
  THIS IMPLEMENTATION IS CORRECT AS THE WAY I SEPARATED MY FUNCTIONS,
I WAS ABLE TO PUT IN AN INVALID HEIGHT FOR A RECTANGLE IN A GROUP AND
THIS FUNCTION WAS ABLE TO CATCH IT AND RETURNED FALSE.
}
```

```
xmlDoc *svgToXmlDoc(const SVG *img) {
  MAKE SURE SVG IS NOT NULL AS SPECIFIED BY SVGParser.h
  if (img == NULL) {
    return NULL;
  }
  // add basic properties like version, type, and namespace
  // need to typecast xmlChar for xml methods
  ADDING MAIN PROPERTIES SUCH AS VERSION, SVG, AND NAMESPACE
  xmlDoc *imgXML = xmlNewDoc((xmlChar*)"1.0");
  xmlNode *root = xmlNewNode(NULL, (xmlChar*)"svg");
  // first use xmlnewns to create the namespace
  // then use xmlsetns to set the namespace for root node
  xmlNs *namespace = xmlNewNs(root, (xmlChar*)img->namespace, NULL);
  xmlSetNs(root, namespace);
  xmlDocSetRootElement(imgXML, root);
  // now do title and description but check for them first
  ONLY ADD TITLE AND DESCRIPTION IF THEY EXIST IN THE IMAGE
  if (strlen(img->title) > 0) {
    xmlNode *imgTitle = xmlNewNode(xmlDocGetRootElement(imgXML)->ns,
(xmlChar*)"title");
    xmlNodeSetContent(imgTitle, (xmlChar*)img->title);
    xmlAddChild(xmlDocGetRootElement(imgXML), imgTitle);
  }
  if (strlen(img->description) > 0) {
    xmlNode *imgDesc = xmlNewNode(xmlDocGetRootElement(imgXML)->ns,
(xmlChar*)"desc");
    xmlNodeSetContent(imgDesc, (xmlChar*)img->description);
    xmlAddChild(xmlDocGetRootElement(imgXML), imgDesc);
  }
  // helper functions to add each shape/attr in the right order
  ADDING SHAPES ONLY IF THEY EXIST IN THE IMAGE USING HELPER FUNCTIONS
FOR EACH (EXPLAINED MORE BELOW)
  if (img->otherAttributes->length > 0) {
    addAttributesXML(img->otherAttributes,
xmlDocGetRootElement(imgXML));
  }
  if (img->rectangles->length > 0) {
    addRectsXML(img->rectangles, xmlDocGetRootElement(imgXML));
  }
  if (img->circles->length > 0) {
    addCirclesXML(img->circles, xmlDocGetRootElement(imgXML));
  }
  if (img->paths->length > 0) {
    addPathsXML(img->paths, xmlDocGetRootElement(imgXML));
  }
  if (img->groups->length > 0) {
    addGroupsXML(img->groups, xmlDocGetRootElement(imgXML));
  }
```

```
  return imgXML;
  THIS IMPLEMENTATION IS CORRECT BECAUSE WE MAKE SURE TO ADD THE
SHAPES IN THE ORDER SPECIFIED IN PDF AND OUR HELPER FUNCTIONS ALLOW
FOR EASY ORGANIZATION AND STRAIGHT-FORWARD IMPLEMENTATION
}

THIS FUNCTION TAKES IN A LIST AND XMLNODE THAT WE SET OUR TEMP TO
void addRectsXML(List *list, xmlNode *node) {
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  Rectangle *r = NULL;
  // iterate until no more rects
  ITERATING THROUGH ALL RECTANGLES IN IMAGE AND MAKING A TEMPNODE FOR
"RECT"
  while ((r = nextElement(&iter)) != NULL) {
    xmlNode* tempNode = xmlNewNode(node->ns, (xmlChar*)"rect");
    // adds all the props to the new xml node
    char *val = malloc(sizeof(char) * 1024);
    NOW WE CAN PRINT THE VALUES AND UNITS TO THE ALLOCATED STRING AND
FOR EACH ONE, USE xmlNewProp() TO ADD THE ELEMENTS AS X, Y, WIDTH, AND
HEIGHT
    sprintf(val, "%f%s", r->x, r->units);
    xmlNewProp(tempNode, (xmlChar*)"x", (xmlChar*)val);
    sprintf(val, "%f%s", r->y, r->units);
    xmlNewProp(tempNode, (xmlChar*)"y", (xmlChar*)val);
    sprintf(val, "%f%s", r->width, r->units);
    xmlNewProp(tempNode, (xmlChar*)"width", (xmlChar*)val);
    sprintf(val, "%f%s", r->height, r->units);
    xmlNewProp(tempNode, (xmlChar*)"height", (xmlChar*)val);
    WE USE OUR ATTRIBUTE XML FUNCTION TO ADD THE OTHER ATTRIBUTE LIST
OF RECTANGLES TO OUR NODE
    addAttributesXML(r->otherAttributes, tempNode);
    // adds new node to svg doc
    FINALLY ADD OUR NODE AND FREE MEMORY
    xmlAddChild(node, tempNode);
    free(val);
  }
}
```

DO THE SAME AS ABOVE TO THE REST OF THE SHAPE LISTS WITH THE ONLY DIFFERENCE BEING THE PROPERTIES OF THE CORRESPONDING SHAPE

```
void addAttributesXML(List *list, xmlNode *node) {
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  Attribute *attr = NULL;
  // traverse until attr is null
  while ((attr = nextElement(&iter)) != NULL) {
    xmlNewProp(node, (xmlChar*)attr->name, (xmlChar*)attr->value);
```

```
  }
}

void addCirclesXML(List *list, xmlNode *node) {
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  Circle *c = NULL;
  // traverse until circles are null
  while ((c = nextElement(&iter)) != NULL) {
    xmlNode* tempNode = xmlNewNode(node->ns, (xmlChar*)"circle");
    // adds all the props to the new xml node
    char *val = malloc(sizeof(char) * 1024);
    sprintf(val, "%f%s", c->cx, c->units);
    xmlNewProp(tempNode, (xmlChar*)"cx", (xmlChar*)val);
    sprintf(val, "%f%s", c->cy, c->units);
    xmlNewProp(tempNode, (xmlChar*)"cy", (xmlChar*)val);
    sprintf(val, "%f%s", c->r, c->units);
    xmlNewProp(tempNode, (xmlChar*)"r", (xmlChar*)val);
    addAttributesXML(c->otherAttributes, tempNode);
    // adds new node to svg doc
    xmlAddChild(node, tempNode);
    free(val);
  }
}

void addPathsXML(List *list, xmlNode *node) {
  // from LinkedListAPI.c
  ListIterator iter = createIterator(list);
  Path *p = NULL;
  // traverse until no more paths
  while ((p = nextElement(&iter)) != NULL) {
    xmlNode* tempNode = xmlNewNode(node->ns, (xmlChar*)"path");
    // adds all the props to the new xml node
    char *val = malloc(sizeof(char) * (strlen(p->data) + 1000000));
    sprintf(val, "%s", p->data);
    xmlNewProp(tempNode, (xmlChar*)"d", (xmlChar*)val);
    addAttributesXML(p->otherAttributes, tempNode);
    // adds new node to svg doc
    xmlAddChild(node, tempNode);
    free(val);
  }
}

void addGroupsXML(List *list, xmlNode *node) {
  // from LinkedListAPi.c
  ListIterator iter = createIterator(list);
  Group *g = NULL;
  // traverse through until no more groups
  while ((g = nextElement(&iter)) != NULL) {
```

```
    xmlNode *tempNode = xmlNewNode(node->ns, (xmlChar*)"g");
    // adds all the props to the new xml node
    addAttributesXML(g->otherAttributes, tempNode);
    addRectsXML(g->rectangles, tempNode);
    addCirclesXML(g->circles, tempNode);
    addPathsXML(g->paths, tempNode);
    addGroupsXML(g->groups, tempNode);
    // adds new node to svg doc
    xmlAddChild(node, tempNode);
  }
}
```

THIS FUNCTION IS USED TO CHECK IF THE CONVERTED XML FROM THE IMAGE IS VALID BASED ON THE XSD FILE

```
int isValidXml(xmlDoc *xml, const char *xsdFile) {
  // all xml variables
  THESE ARE USED TO ASSIGN THE PARSER, SCHEMA, AND VALIDATOR
  xmlSchemaParserCtxt *parserContext = NULL;
  xmlSchema *schema = NULL;
  xmlSchemaValidCtxt *validator = NULL;
  int ret = -1;

  FILE *fp = fopen(xsdFile, "r");
  parserContext = xmlSchemaNewParserCtxt(xsdFile);
  schema = xmlSchemaParse(parserContext);
  validator = xmlSchemaNewValidCtxt(schema);

  // all xml validation against schema files
  HERE I MAKE SURE THE XML FILE, SCHEMA FILE, FILE POINTER, PARSER, SCHEMA PARSER, AND VALIDATOR IS NULL. IF SO WE HAVE TO FREE THE XML DOC AND RETURN -1 INDICATING A FAILURE
  if (xml == NULL || xsdFile == NULL || fp == NULL || parserContext == NULL || schema == NULL || validator == NULL) {
    xmlFreeDoc(xml);
  } else {
    OTHERWISE, WE VALIDATE THE DOC AND ASSIGN THE INT. WE ALSO FREE THE MEMORY TAKEN BY OUR PARSER, SCHEMA PARSER, AND VALIDATOR ONLY IF THEY WERE NOT NULL BECAUSE IF THEY WERE, CAN CAUSE MEMORY ERRORS
    ret = xmlSchemaValidateDoc(validator, xml);
    if (parserContext != NULL) {
      xmlSchemaFreeParserCtxt(parserContext);
    }
    if (schema != NULL) {
      xmlSchemaFree(schema);
    }
    if (validator != NULL) {
      xmlSchemaFreeValidCtxt(validator);
    }
  }
```

```
  NOW WE CAN RETURN THE VALUE FROM xmlSchemaValidateDoc()
  fclose(fp);
  return ret;
}
```

2.2 - rectListToJSON()

- Filename: SVGParser.c, Line: 1173-1197, Repo: CIS2750W22_Project, Branch: A2
- **ANNOTATIONS ARE IN THE CODE SNIPPETS BOLDED**

```
char* rectListToJSON(const List *list) {
  // check args first and return empty []
  IF ARGUMENT LIST IS NULL OR LIST IS EMPTY, WE RETURN A STRING THAT
CONTAINS "[]"
  if (list == NULL || list->head == NULL) {
    char *invalidStr = malloc(sizeof(char) * 4);
    strcpy(invalidStr, "[]");
    RETURNING JUST THE CHARS RESULTED IN MEMORY ERRORS SO I DECIDED TO
MALLOC IT INSTEAD AND RETURN THAT WHICH WORKED
    return invalidStr;
  }
  // iterate thru and dynamically allocate space for the string
  ListIterator iter = createIterator((List*)list);
  Rectangle *r = NULL;
  FIRST ALLOCATE INITIAL SPACE FOR THE NEW STRING AND COPY THE
"[" CHARACTER INDICATING THE START OF THE LIST
  char *str = malloc(sizeof(char) + 1024);
  // first char of string
  strcpy(str, "[");
  // now iterate
  ITERATING THROUGH ALL RECTANGLES IN THE LIST AND FOR EACH ONE, WE
CALL rectToJSON() WHICH WILL COPY OVER THE PROPERTIES OF THE RECTANGLE
SUCH AS X, Y, WIDTH, HEIGHT AS WELL AS ALLOWS US TO RE-ALLOCATE MEMORY
BASED ON THE RETURNED STRING DUE TO THE CONSTRAINT WHERE WE MUST NOT
MAKE ASSUMPTIONS ABOUT THE LENGTH OF THE LIST
  while ((r = nextElement(&iter)) != NULL) {
    char *jsonRect = rectToJSON(r);
    SINCE WE ARE ITERATING USING nextElement(), WE MAKE SURE THE ORDER
OF THE STRINGS ARE THE SAME AS THE ORDER OF THE ORIGINAL LIST
    str = realloc(str, strlen(str) + strlen(jsonRect) + 8);
    strcat(str, jsonRect);
    AFTER EACH CONCATENATION OF THE JSON STRING, WE FOLLOW IT WITH A
"," CHARACTER
    strcat(str, ",");
    free(jsonRect);
  }
  WE CANNOT CONCATENATE "]" BY ITSELF TO CLOSE OF THE STRING AS WE
WOULD HAVE AN EXTRA "," CHARACTER SO WE EXPLICITLY CHANGE THE LAST
CHARACTER BEING "," TO A "]" CHARACTER
  // now close of json string
```

```
  str[strlen(str) - 1] = ']';
  return str;
```
THIS IMPLEMENTATION IS CORRECT BECAUSE AFTER USING THE TEST HARNESS, THE EXPECTED STRING WAS OUTPUTTED AND DID NOT FAIL ANY CASE. WE ALSO DO NOT MODIFY THE FUNCTION ARGUMENT IN ANYWAY SINCE WE ALLOCATE A NEW STRING THAT GETS RETURNED NO MATTER IF THE ARGUMENT IS NULL OR NOT
```
}
```

FUNCTION USED FOR rectListToJSON()

```
char* rectToJSON(const Rectangle *r) {
  // check args first and return empty json
```
MAKE SURE IF THE RECTANGLE IS NULL, RETURN AN EMPTY "{}"
```
  if (r == NULL) {
    return "{}";
  }
```
ALLOCATE SPACE FOR THE VALUES AND PRINT THEM TO THE STRING IN THE EXACT ORDER SPECIFIED BY THE FUNCTION SPEC. THIS ALSO MAKES SURE IN rectListToJSON(), WE CAN CALL IT MULTIPLE TIMES IN A LOOP AND PRESERVE THE FORMAT
```
  // allocate space and write in the values
  char *str = malloc((strlen(r->units) + 1000) * sizeof(char));
  sprintf(str, "{\"x\":%.2f,\"y\":%.2f,\"w\":%.2f,\"h\":%.2f,
\"numAttr\":%d,\"units\":\"%s\"}", r->x, r->y, r->width, r->height, r-
>otherAttributes->length, r->units);
  return str;
```
THIS IMPLEMENTATION IS CORRECT BECAUSE I FOLLOWED THE EXACT FORMAT SPECIFIED IN THE PDF AND I DO NOT ADD ANY SPACES OR NEWLINES IN THE PROCESS.
```
}
```