DOKUZ EYLUL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2202 DATA ORGANIZATION AND MANAGEMENT

REPORT

ASSIGNMENT - I

-File Convertor - Bin2XML-

By

Merve Doğan 2019510028

Nadirhan Şahin 2019510130

Lecturers

Asst.Prof.Dr. Özlem AKTAŞ

Res.Asst.Dr. Meltem YILDIRIM EKİCİ

Res.Asst.Dr. İbrahim Atakan KUBİLAY

24 April 2022

İZMİR

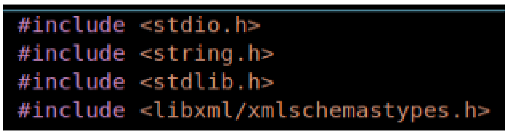
CHAPTER ONE

DETAIL EXPLANATION OF OUR SOLUTION

- We created a struct called "record" to read and save data from the "records.dat" file.
- Then we created a "record" array called persons. In this way, we were able to keep all the data in an array.
- First, we took the first sample of the "records.dat" file as a tag, then replaced the wrongly written tags with the correct ones.
- We did this operation using "!strcmp(tagname, "tagname") ? 0 : strcpy(tagname, "tagname")" code snipset.
- Using "feof(filepointer)" we checked whether we reached the end of the file while reading the data.
- In the while loop, we saved the data to the "record" struct one by one and sent it to the "persons" array.
- Then, we printed the "persons" array, which we hold the data, into XML in the for loop (using the "counter" variable we created earlier before the while loop).
- We used the "fromUtf16" and "fromUtf8" functions while printing to XML.
- We first decoded the incoming texts with these functions and returned them to Unicode values (we kept them as 32-bit unsigned integers).
- Then we encoded them with the "toUtf8" and "toUtf16" functions and restored them to their original state.
- After these transformations, we selected the tags we wanted and printed them in XML.
- We have determined the place where the XML will be created as the 2nd of the arguments received from the user. (the first was for the path of the "records.dat" file)
- We created an XSD file by selecting the attributes we wanted.
- In the next steps, we validated the XML file with the XSD file. (We took the path of the XSD file as the 3rd argument)
- We used the "libxml2" library for the validation process.
  - In this library:*We used functions "xmlReadFile", "xmlSchemaSetParserErrors", "xmlSchemaNewValidCtxt", "xmlSchemaSetValidErrors", "xmlSchemaValidateDoc", "xmlSchemaFreeValidCtxt", "xmlFreeDoc". Thus we got a validated XML file.*
- To get the Unicode values, functions("fromUtf16" and "fromUtf8"):
  - Accepts a word as an argument
  - Checks with if else if the binary values of these characters exceed certain thresholds (in for loop)
  - Determines how many bytes will be used according to these threshold values
  - Saves each byte into a char array (byte[0], byte[1]) with "&" operation
  - For example, the char "ş" holds 2x8 bits, that is, 2 bytes.
  - Sends this data to function "toUtf8" or "toUtf16"
  - Then saves the data returned from the function to the char array named "word" with the "strcat" function.
  - Thus, all the chars are saved consecutively.
- To revert from Unicode values functions ("toUtf16" and "toUtf8"):
  - Accepts a maximum of 4 bytes of char arrays as arguments. (equivalent to 1 char)
  - The reverse is done in the "fromUtf16" and "fromUtf8" functions of the char array that comes as an argument.
  - Thus, it encodes the character

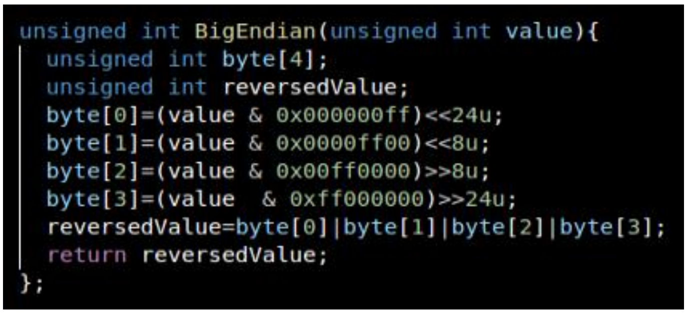
CHAPTER TWO

SAMPLE SCREENSHOTS



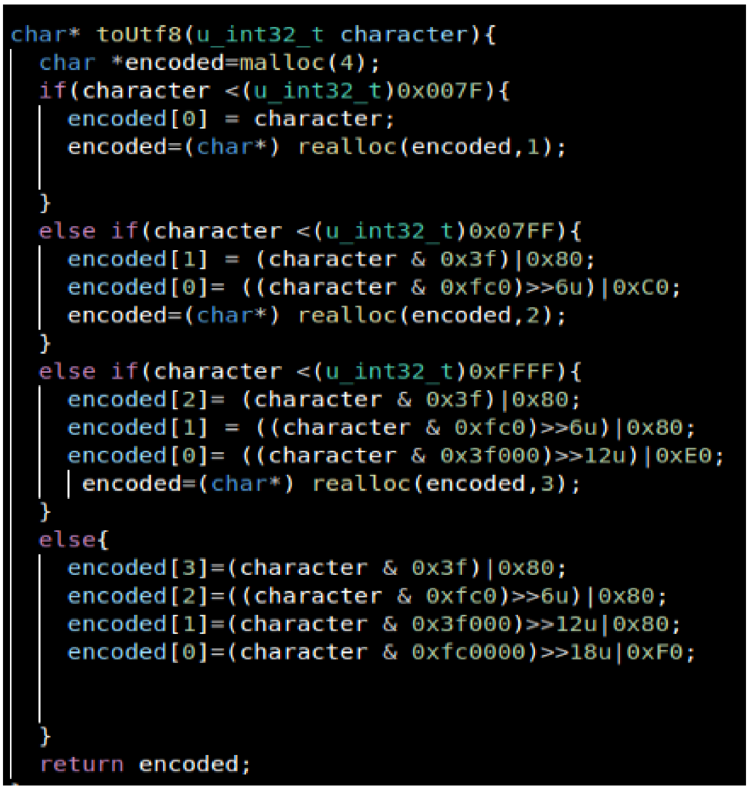
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlschemastypes.h>
```

libraries we use in homework



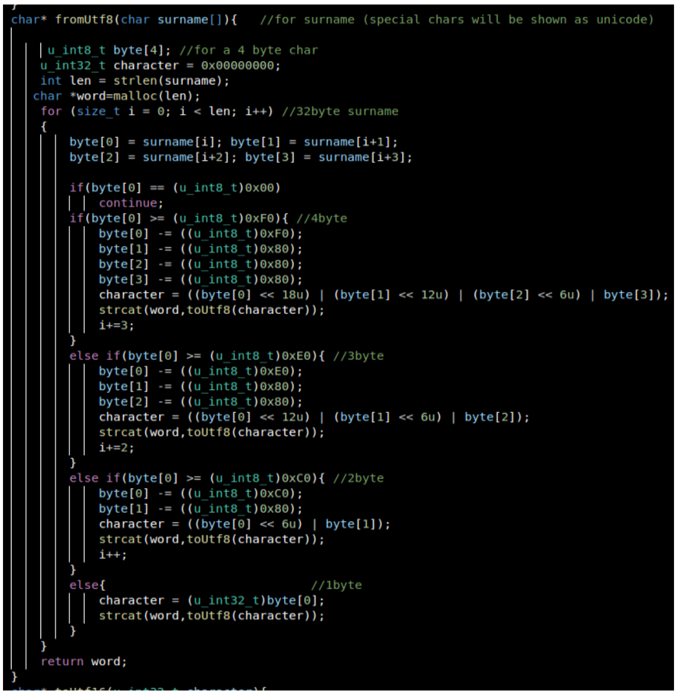
```
unsigned int BigEndian(unsigned int value){
   unsigned int byte[4];
   unsigned int reversedValue;
   byte[0]=(value & 0x000000ff)<<24u;
   byte[1]=(value & 0x0000ff00)<<8u;
   byte[2]=(value & 0x00ff0000)>>8u;
   byte[3]=(value  & 0xff000000)>>24u;
   reversedValue=byte[0]|byte[1]|byte[2]|byte[3];
   return reversedValue;
};
```

'BigEndian' function



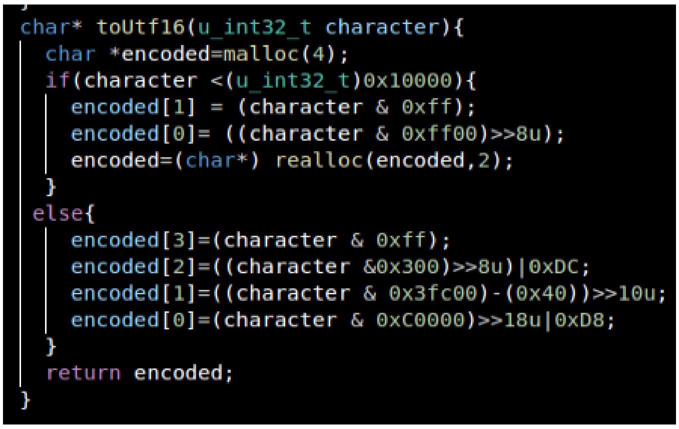
```
char* toUtf8(u_int32_t character){
   char *encoded=malloc(4);
   if(character <(u_int32_t)0x007F){
      encoded[0] = character;
      encoded=(char*) realloc(encoded,1);

   }
   else if(character <(u_int32_t)0x07FF){
      encoded[1] = (character & 0x3f)|0x80;
      encoded[0]= ((character & 0xfc0)>>6u)|0xC0;
      encoded=(char*) realloc(encoded,2);
   }
   else if(character <(u_int32_t)0xFFFF){
      encoded[2]= (character & 0x3f)|0x80;
      encoded[1] = ((character & 0xfc0)>>6u)|0x80;
      encoded[0]= ((character & 0x3f000)>>12u)|0xE0;
        encoded=(char*) realloc(encoded,3);
   }
   else{
      encoded[3]=(character & 0x3f)|0x80;
      encoded[2]=((character & 0xfc0)>>6u)|0x80;
      encoded[1]=(character & 0x3f000)>>12u|0x80;
      encoded[0]=(character & 0xfc0000)>>18u|0xF0;


   }
   return encoded;
```

'toUtf8' function



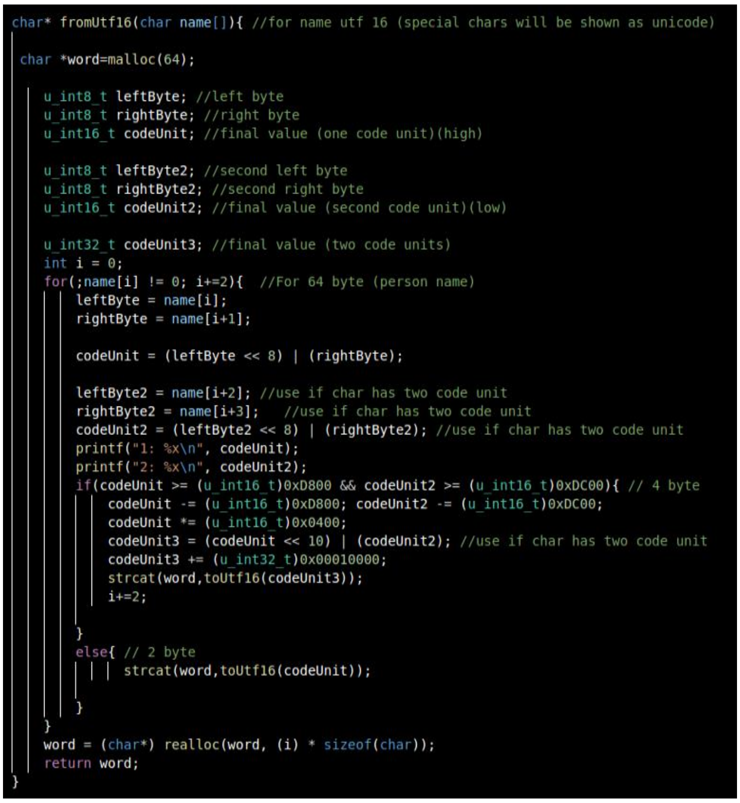
```c
char* fromUtf8(char surname[]){   //for surname (special chars will be shown as unicode)

     u_int8_t byte[4]; //for a 4 byte char
    u_int32_t character = 0x00000000;
    int len = strlen(surname);
   char *word=malloc(len);
    for (size_t i = 0; i < len; i++) //32byte surname
    {
        byte[0] = surname[i]; byte[1] = surname[i+1];
        byte[2] = surname[i+2]; byte[3] = surname[i+3];

        if(byte[0] == (u_int8_t)0x00)
            continue;
        if(byte[0] >= (u_int8_t)0xF0){ //4byte
            byte[0] -= ((u_int8_t)0xF0);
            byte[1] -= ((u_int8_t)0x80);
            byte[2] -= ((u_int8_t)0x80);
            byte[3] -= ((u_int8_t)0x80);
            character = ((byte[0] << 18u) | (byte[1] << 12u) | (byte[2] << 6u) | byte[3]);
            strcat(word,toUtf8(character));
            i+=3;
        }
        else if(byte[0] >= (u_int8_t)0xE0){ //3byte
            byte[0] -= ((u_int8_t)0xE0);
            byte[1] -= ((u_int8_t)0x80);
            byte[2] -= ((u_int8_t)0x80);
            character = ((byte[0] << 12u) | (byte[1] << 6u) | byte[2]);
            strcat(word,toUtf8(character));
            i+=2;
        }
        else if(byte[0] >= (u_int8_t)0xC0){ //2byte
            byte[0] -= ((u_int8_t)0xC0);
            byte[1] -= ((u_int8_t)0x80);
            character = ((byte[0] << 6u) | byte[1]);
            strcat(word,toUtf8(character));
            i++;
        }
        else{                            //1byte
            character = (u_int32_t)byte[0];
            strcat(word,toUtf8(character));
        }
    }
    return word;
}
```

'fromUtf8' function



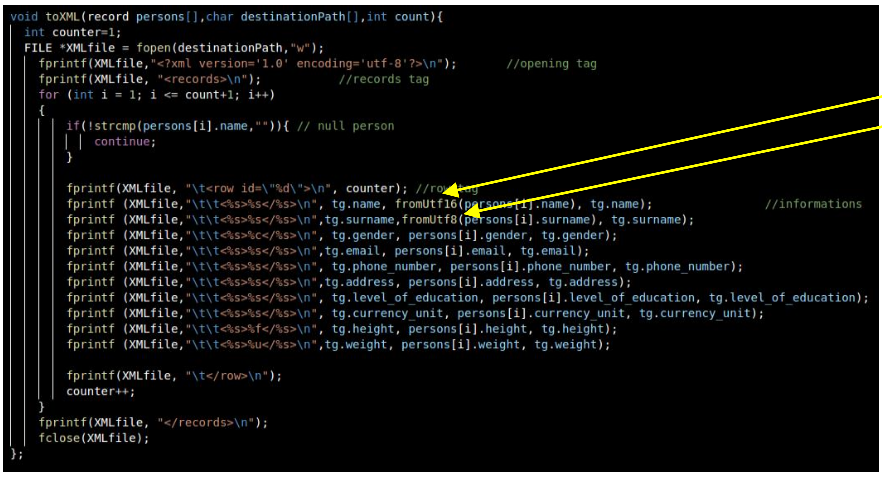
```c
char* toUtf16(u_int32_t character){
  char *encoded=malloc(4);
  if(character <(u_int32_t)0x10000){
    encoded[1] = (character & 0xff);
    encoded[0]= ((character & 0xff00)>>8u);
    encoded=(char*) realloc(encoded,2);
  }
 else{
    encoded[3]=(character & 0xff);
    encoded[2]=((character &0x300)>>8u)|0xDC;
    encoded[1]=((character & 0x3fc00)-(0x40))>>10u;
    encoded[0]=(character & 0xC0000)>>18u|0xD8;
  }
  return encoded;
}
```

'toUtf16' function



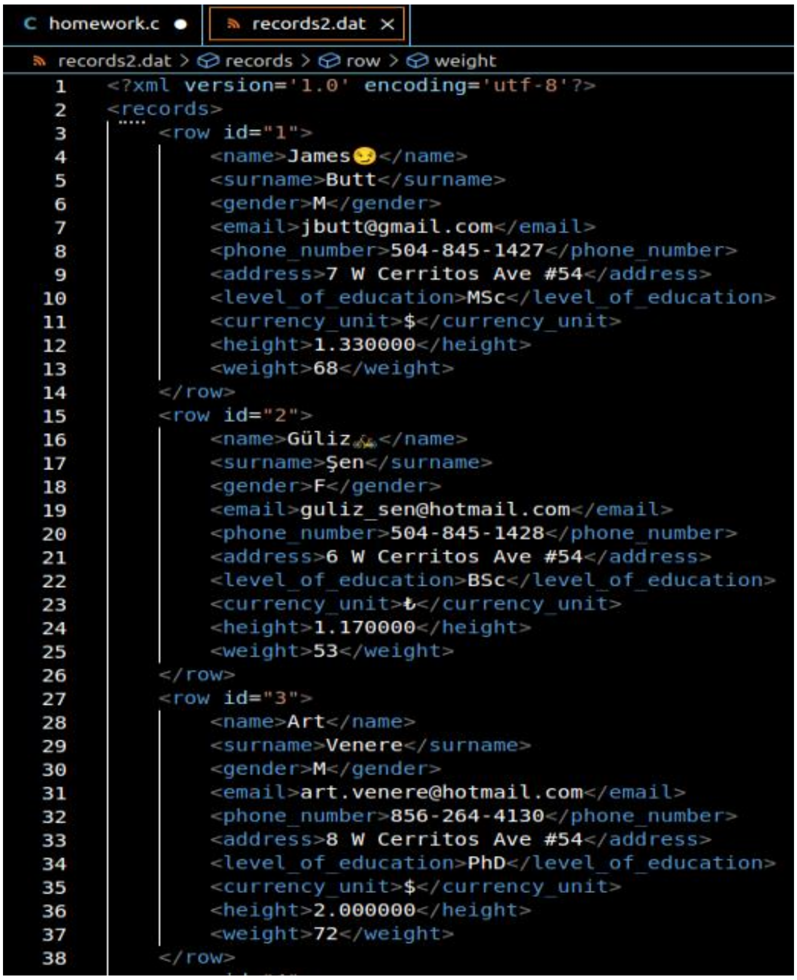
```c
char* fromUtf16(char name[]){ //for name utf 16 (special chars will be shown as unicode)

 char *word=malloc(64);

   u_int8_t leftByte; //left byte
   u_int8_t rightByte; //right byte
   u_int16_t codeUnit; //final value (one code unit)(high)

   u_int8_t leftByte2; //second left byte
   u_int8_t rightByte2; //second right byte
   u_int16_t codeUnit2; //final value (second code unit)(low)

   u_int32_t codeUnit3; //final value (two code units)
   int i = 0;
   for(;name[i] != 0; i+=2){  //For 64 byte (person name)
       leftByte = name[i];
       rightByte = name[i+1];

       codeUnit = (leftByte << 8) | (rightByte);

       leftByte2 = name[i+2]; //use if char has two code unit
       rightByte2 = name[i+3];   //use if char has two code unit
       codeUnit2 = (leftByte2 << 8) | (rightByte2); //use if char has two code unit
       printf("1: %x\n", codeUnit);
       printf("2: %x\n", codeUnit2);
       if(codeUnit >= (u_int16_t)0xD800 && codeUnit2 >= (u_int16_t)0xDC00){ // 4 byte
           codeUnit -= (u_int16_t)0xD800; codeUnit2 -= (u_int16_t)0xDC00;
           codeUnit *= (u_int16_t)0x0400;
           codeUnit3 = (codeUnit << 10) | (codeUnit2); //use if char has two code unit
           codeUnit3 += (u_int32_t)0x00010000;
           strcat(word,toUtf16(codeUnit3));
           i+=2;

       }
       else{ // 2 byte
             strcat(word,toUtf16(codeUnit));

       }
   }
   word = (char*) realloc(word, (i) * sizeof(char));
   return word;
}
```

'fromUtf16' function



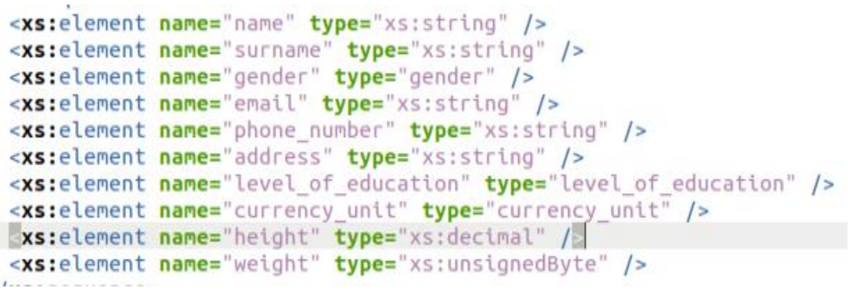
```c
void toXML(record persons[],char destinationPath[],int count){
  int counter=1;
  FILE *XMLfile = fopen(destinationPath,"w");
    fprintf(XMLfile,"<?xml version='1.0' encoding='utf-8'?>\n");       //opening tag
    fprintf(XMLfile, "<records>\n");          //records tag
    for (int i = 1; i <= count+1; i++)
    {
        if(!strcmp(persons[i].name,"")){ // null person
            continue;
        }

        fprintf(XMLfile, "\t<row id=\"%d\">\n", counter); //row tag
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n", tg.name, fromUtf16(persons[i].name), tg.name);             //informations
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n",tg.surname,fromUtf8(persons[i].surname), tg.surname);
        fprintf (XMLfile,"\t\t<%s>%c</%s>\n", tg.gender, persons[i].gender, tg.gender);
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n",tg.email, persons[i].email, tg.email);
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n", tg.phone_number, persons[i].phone_number, tg.phone_number);
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n",tg.address, persons[i].address, tg.address);
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n", tg.level_of_education, persons[i].level_of_education, tg.level_of_education);
        fprintf (XMLfile,"\t\t<%s>%s</%s>\n", tg.currency_unit, persons[i].currency_unit, tg.currency_unit);
        fprintf (XMLfile,"\t\t<%s>%f</%s>\n", tg.height, persons[i].height, tg.height);
        fprintf (XMLfile,"\t\t<%s>%u</%s>\n",tg.weight, persons[i].weight, tg.weight);

        fprintf(XMLfile, "\t</row>\n");
        counter++;
    }
    fprintf(XMLfile, "</records>\n");
    fclose(XMLfile);
};
```

‘toXML’ function to write in XML. In these yellow arrows we call our ‘fromUtf8’ and ‘fromUtf16’ functions to print in the desired format



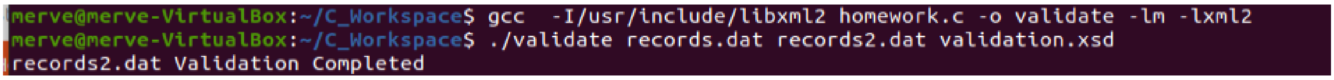
```xml
<?xml version='1.0' encoding='utf-8'?>
<records>
    <row id="1">
        <name>James😏</name>
        <surname>Butt</surname>
        <gender>M</gender>
        <email>jbutt@gmail.com</email>
        <phone_number>504-845-1427</phone_number>
        <address>7 W Cerritos Ave #54</address>
        <level_of_education>MSc</level_of_education>
        <currency_unit>$</currency_unit>
        <height>1.330000</height>
        <weight>68</weight>
    </row>
    <row id="2">
        <name>Güliz🚴</name>
        <surname>Şen</surname>
        <gender>F</gender>
        <email>guliz_sen@hotmail.com</email>
        <phone_number>504-845-1428</phone_number>
        <address>6 W Cerritos Ave #54</address>
        <level_of_education>BSc</level_of_education>
        <currency_unit>₺</currency_unit>
        <height>1.170000</height>
        <weight>53</weight>
    </row>
    <row id="3">
        <name>Art</name>
        <surname>Venere</surname>
        <gender>M</gender>
        <email>art.venere@hotmail.com</email>
        <phone_number>856-264-4130</phone_number>
        <address>8 W Cerritos Ave #54</address>
        <level_of_education>PhD</level_of_education>
        <currency_unit>$</currency_unit>
        <height>2.000000</height>
        <weight>72</weight>
    </row>
```

Xml file(records2.dat) after reading from records.dat file



```xml
<xs:element name="name" type="xs:string" />
<xs:element name="surname" type="xs:string" />
<xs:element name="gender" type="gender" />
<xs:element name="email" type="xs:string" />
<xs:element name="phone_number" type="xs:string" />
<xs:element name="address" type="xs:string" />
<xs:element name="level_of_education" type="level_of_education" />
<xs:element name="currency_unit" type="currency_unit" />
<xs:element name="height" type="xs:decimal" />
<xs:element name="weight" type="xs:unsignedByte" />
```

Part of the validation.xsd file that we created to validate our xml file



```
merve@merve-VirtualBox:~/C_Workspace$ gcc  -I/usr/include/libxml2 homework.c -o validate -lm -lxml2
merve@merve-VirtualBox:~/C_Workspace$ ./validate records.dat records2.dat validation.xsd
records2.dat Validation Completed
```

CHAPTER THREE

PROBLEMS ENCOUNTERED

- ➔ Since we called Utf8 and utf16 ready for file reads, we had no idea what kind of reading they were doing. So initially we investigated how the functions utf8 and utf16 store words and how words are returned from utf8-utf16. However, the lack of resources on this subject made it difficult for us to understand and caused us to spend a lot of time on resource research.

- ➔ When writing utf16 functions, we didn't have much trouble keeping the unicode equivalent of words, but we had a hard time returning these unicode counterparts from utf16. Since the computer separates words that take up more than one byte into more than one char, we combined the hex codes with the strcat function and reached the hex char of a character and handled the return operations.

- ➔ In the XSD validation part, we had difficulties in importing the libxml library and using the library. Since there were not enough resources, we were able to use the library by reading how people use the library from the forums and handled the xsd validation part.