COMPUTER ORGANIZATION FINAL PROJECT

MOST FUN PART

- learning how other languages can interpreted by computers
- coming up with our own functions

HARDEST PART

- encode and decode
- The inputs and what form they should be in
- binary to hex and hex to binary

HEBREW PLURALIZER

```
char *my_utf8_hebrewpluralizer(char *string){
    //find the len of the str
    int a = my_utf8_check(string);
    if (a == 0){
        return NULL;
    }
    int i = 0;
    int strlen = 0;
    while (string[i] != '\0'){
        if (!((unsigned char)string[i] == 0xD7)){
        return NULL;
        }
        strlen++;
        i += 2;
    }
    //look at the last letter
    char lastLetter[3];
    lastLetter[0] = string[i - 2];
    lastLetter[1] = string[i - 1];
    lastLetter[2] = '\0';
    char *resultStr = (char*)malloc(i + 4);
    // if letter ends with ה or ת
    if((unsigned char)lastLetter[1] ==  0x94 ||(unsigned char)lastLetter[1] == 0xAA){
        for(int j = 0; j < i - 2; j++){
            resultStr[j] = string[j];
        }
        resultStr[i - 2] = 0xD7;
        resultStr[i-1] = 0x95;
        resultStr[i] = 0xD7;
        resultStr[i + 1] = 0xAA;
        resultStr[i + 2] = '\0';
    }
else{
      for(int j = 0; j < i - 1; j++){
          resultStr[j] = string[j];
       }
       // if ם then switch to מ
       if ((unsigned char)string[i] == 0x9D){
          resultStr[i] = 0x9E;
       }
       //if ן then switch to נ
       else if ((unsigned char)string[i] == 0x9F){
          resultStr[i] = 0xA0;
       }
       //if ך then sqwitch to כ
       else if ((unsigned char)string[i] == 0x9A){
          resultStr[i] = 0x9B;
       }
       // if ף then switch to פ
       else if ((unsigned char)string[i] == 0xA3){
          resultStr[i] = 0xA4;
       }
       // if ץ then switch to צ
       else if((unsigned char)string[i] == 0xA5){
          resultStr[i] = 0xA6;
       }
       //if regular letter
       else{
          resultStr[i] = string[i];
       }
       //change the last 2 letters to ם'
       resultStr[i] = 0xD7;
       resultStr[i + 1] = 0x99;
       resultStr[i + 2] = 0xD7;
       resultStr[i + 3] = 0x9D;
       resultStr[i + 4] = '\0';
    }
    return resultStr;
```

REVERSE STRING

```
char *my_utf8_strreverse(char *string){
    //find the len of the str
    int a = my_utf8_check(string);
    if (a == 0){
        return NULL;
    }
    int i = 0;
    int strlen = 0;
    //copy strlen function
    while (string[i] != '\0'){
        if ((unsigned char)string[i] <= 0x7F){
            strlen++;
        }
        else if ((unsigned char)string[i] >= 0xC0 && (unsigned char)string[i] <= 0xDF){
            strlen++;
            i++;
        }
        else if((unsigned char)string[i] >= 0xE0 && (unsigned char)string[i] <= 0xEF){
            strlen++;
            i+=2;
        }
        else if((unsigned char)string[i] >= 0xF0 && (unsigned char)string[i] <= 0xF7){
            strlen++;
            i+=3;
        }
        i++;
    }
    if (strlen == 0){
        return NULL;
    }

    //allocate the memory needed for the reverse str
    char *resultStr = (char*)malloc(i + 1);
    //fill the resultStr by putting them in backwards
    int resultIndex = 0;
    for (int j = i - 1; j >= 0; j--){
        if ((unsigned char)string[j] <= 0x7F){
            resultStr[resultIndex] = string[j];
            resultIndex++;
        }
        else if ((unsigned char)string[j - 1] >= 0xC0 && (unsigned char)string[j - 1] <= 0xDF){
            resultStr[resultIndex] = string[j - 1];
            resultStr[resultIndex + 1] = string[j];
            resultIndex += 2;
            j--;
        }
        else if((unsigned char)string[j - 2] >= 0xE0 && (unsigned char)string[j - 2] <= 0xEF){
            resultStr[resultIndex] = string[j - 2];
            resultStr[resultIndex + 1] = string[j - 1];
            resultStr[resultIndex + 2] = string[j];
            j -= 2;
            resultIndex += 3;
        }
        else if((unsigned char)string[j - 3] >= 0xF0 && (unsigned char)string[j - 3] <= 0xF7){
            resultStr[resultIndex] = string[j - 3];
            resultStr[resultIndex] = string[j - 2];
            resultStr[resultIndex] = string[j - 1];
            resultStr[resultIndex] = string[j];
            resultIndex += 4;
            j -= 3;
        }
    }
    resultStr[i + 1] = '\0';
    return resultStr;
}
```

WHAT DID I LEARN?

- learning all about UTF-8
- I appreciated that we had to learn UTF8 on our own

IF I WAS DOING THIS PROJECT AGAIN I WOULD...

- left my old work for encode and decode
- tested more
-

UTF8- CHECK

```
int my_utf8_check(char *string){
    int i = 0;
    //loop through
    while ((unsigned char)string[i] != '\0'){
        //if in ascii range then its good
        if ((unsigned char)string[i] <= 0x7F){
        }
        // if 2 bytes and the second byte isnt in the range then return -1
        else if ((unsigned char)string[i] >= 0xc0 && (unsigned char)string[i] <= 0xdf){
            if (!((unsigned char)string[i+1] >= 0x80 && (unsigned char)string[i+1] <= 0xBF)){
                return 0;
            }
            i++;
        }
        //if 3 bytes and 2nd isnt in range or third return -1
        else if((unsigned char)string[i] >= 0xE0 && (unsigned char)string[i] <= 0xEF){
            if (!((unsigned char)string[i+1] >= 0x80 && (unsigned char)string[i+1] <= 0xBF)){
                return 0;
            }
            else if (!((unsigned char)string[i+2] >= 0x80 && (unsigned char)string[i+2] <= 0xBF)){
                return 0;
            }
            i+=2;
        }
/
```

```
/if 4 bytes and 2nd isnt in range or third or fourth return -1
 else if((unsigned char)string[i] >= 0xF0 && (unsigned char)string[i] <= 0xF7){
 if (!((unsigned char)string[i+1] >= 0x80 && (unsigned char)string[i+1] <= 0xBF)){
 return 0;
 }
 else if (!((unsigned char)string[i+2] >= 0x80 && (unsigned char)string[i+2] <= 0xBF)){
 return 0;
 }
 else if (!((unsigned char)string[i+3] >= 0x80 && (unsigned char)string[i+3] <= 0xBF)){
 return 0;
 }
 i+=3;
 }
 else{
 return 0;
 }
 i++;
 }
 return 1;
}
```

CHARAT

```
char *my_utf8_charat(char *string, int index){
    int i = 0;
    int strlen = 0;
    // as long as were not an the index
    while (strlen != index){
        //copy the strlen function
        if ((unsigned char)string[i] <= 0x7F){
            strlen++;
        }
        // if 2 bytes and the second byte isnt in the range then return -1
        else if ((unsigned char)string[i] >= 0xC0 && (unsigned char)string[i] <= 0xDF){
            strlen++;
            i++;
        }
        //if 3 bytes and 2nd isnt in range or third return -1
        else if((unsigned char)string[i] >= 0xE0 && (unsigned char)string[i] <= 0xEF){
            strlen++;
            i+=2;
        }
        //if 4 bytes and 2nd isnt in range or third or fourth return -1
        else if((unsigned char)string[i] >= 0xF0 && (unsigned char)string[i] <= 0xF7){
            strlen++;
            i+=3;
        }
        i++;
    }
```

```
// look at how many bytes the letter is and reserve space and then put it in that resultStr
    if ((unsigned char)string[i] <= 0x7F){
        char *resultStr = (char*)malloc(2);
        resultStr[0] = string[i];
        resultStr[1] = '\0';
        return resultStr;
    }
    else if ((unsigned char)string[i] >= 0xC0 && (unsigned char)string[i] <= 0xDF){
        char *resultStr = (char*)malloc(3);
        resultStr[0] = string[i];
        resultStr[1] = string[i + 1];
        resultStr[2] = '\0';
        return resultStr;
    }
    else if((unsigned char)string[i] >= 0xE0 && (unsigned char)string[i] <= 0xEF){
        char *resultStr = (char*)malloc(4);
        resultStr[0] = string[i];
        resultStr[1] = string[i + 1];
        resultStr[2] = string[i + 2];
        resultStr[3] = '\0';
        return resultStr;
    }
    else if((unsigned char)string[i] >= 0xF0 && (unsigned char)string[i] <= 0xF7){
        char *resultStr = (char*)malloc(5);
        resultStr[0] = string[i];
        resultStr[1] = string[i + 1];
        resultStr[2] = string[i + 2];
        resultStr[3] = string[i + 3];
        resultStr[4] = '\0';
        return resultStr;
    }
    return string;
}
```

STRING CMP

```
int my_utf8_strcmp(char *string1, char *string2){
    int i = 0;
    while (string1[i] != '\0' && string2[i] != '\0'){
        if (string1[i] < string2[i]){
            //str1 is less than str2
            return -1;
        }
        else if (string1[i] > string2[i]){
            //str 1 is greater than str2
            return 1;
        }
        string1 ++;
        string2 ++;
    }
    if (string1[i] == '\0' && string2[i] == '\0'){
        return 0;
    }
    else if (string1[i] == '\0'){
        //str1 is shorter
        return -1;
    }
    else if (string2[i] == '\0'){
        return 1;
    }
    return -2;
}
```