Q1: 15 points

Write 2 functions named `prettyprinter` and `ppuser`. These functions have the following documentation at the function level:

```
/*
** prettyprinter
** Args:
**     char* tgt – a string to be printed
**     int  num  – the number of spaces to print in
**     char w – a padding character
** This function prints tgt using only putchar.
** After printing tgt, it repeatedly prints the character in w
** until the total number of chars printed is num. It tgt is
** longer than num
** then truncate tgt to num spaces and print 0 pad chars

**/

/** ppuser
** Args:
**     The first three args are identical to those of
**           prettyprinter
**     The fourth arg is a a function pointer
**           to a function that follows
**           the specification of prettyprinter
** This function contains a single line of code.
** It just calls the provided function pointer with
** the other 3 parameters provided to this function.
**/
```

```
#include<stdio.h>
#include<string.h>

void prettyprinter(char* tgt, int num, char w) {
    int ll = strlen(tgt);
    int i=0;
    for (; i<ll && i<num; i++)
        putchar(tgt[i]);
    for (; i<num; i++)
    {
        putchar(w);
    }
}

void ppuser(char* tgt, int num, char w, void(*pp)(char*, int, char)) {
    pp(tgt, num, w);
}

int main(void) {
    ppuser("this is a test", 20, 'A', prettyprinter);
    printf("\n");
    ppuser("this is a test", 10, 'B', prettyprinter);
    printf("\n");
}
```

Q2: 20 Points

Write a function with the signature:

char* stringreplace(char* src, char* replace, char* replacement)

This function returns a new string in which every instance of the parameter replace will be changed to the parameter replacement. You may assume that all args are properly null terminated. The return value must also be null terminated. replace and replacement may have different lengths. The returned string should have no wasted space.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
// test if tgt (starting at character ti) matches rep
int mtch(int ti, char* tgt, char* rep) {
    for (int i=0; i<strlen(rep) && (ti+i)<strlen(tgt); i++) {
    if (tgt[ti+i]!=rep[i])
        return 0;
    }
    return i==strlen(rep); // make sure that whole of rep matched
}

char* stringreplace(char* tgt, char* rep, char* ment) {
    char tmp[strlen(tgt)*strlen(rep)];
    int ii=0; int jj=0;
    while (ii<strlen(tgt)) {
    if (mtch(ii, tgt, rep))
    {
        for (int i3=0; i3<strlen(ment); i3++)
        tmp[jj++]=ment[i3];
        ii+=strlen(rep);
    }
    else {
        tmp[jj++]=tgt[ii++];
    }  }
    char* news = malloc((jj+1)*sizeof(char));
    for (int i=0; i<jj; i++)
        news[i]=tmp[i];
    news[jj+1]='\0';
    return news;
}

int main(void) {
    printf("||%s||\n", stringreplace("this is a test", "is", "was"));
}
```

Q3: 15 Points

Write regular expressions that will do the following (for instance when used with egrep)

a. find all lines that contain only the word right preceded by exactly 2 characters or the word copy. That is, this should find lines with “beright”, and “copyright” but not lines containing “right”, “aright”, “heright”, etc (unless the line aslo contains “copyright”, …)

```
[[:blank::]](copy | .{2})right[[:blank:]]
```

b. find all lines that contain both ‘abc’ and ‘xyz’, in either order

```
(abc.*xyz|xyz.*abc)
```

c. find all lines containing at least one word 12 to 15 characters

```
[[:blank]][[:alpha:]]{12,15}[[:blank:]]
```

d. find all lines containing a word of the form a vowel followed by 0 or 1 nonvowels, followed by a vowel followed by 1 or more nonvowels. For instance, “ear” and “ewer” both would be matched by this regexp.

```
[[:blank:]][aeiouAEIOU][^aeiouAEIOU]?[aeiouAEIOU][^aeiouAEIOU]+
[[:blank:]]
```

e. find all lines that contain strings that start with st end with y and do not have any numbers between

```
^st[^1-9]*y$
```

Q4: 10 Points

In a store, every item has a name (a string of less than 50 characters), UPC number (a string of exactly 22 characters), and location. The location may be given by either a row number and bin number or a string describing a general location (e.g., "front"). You many assume that this string has no more than 20 characters. Define a data structure that efficiently captures this information.

```
typdef struct {
	int row;
	int bin;
} RB;

typedef union {
	RB rb;
	char name[20];
} Location;

typedef struct {
	char name[50];
	char UPC[23];
	Location location;
} Item;
```

Q5: 10 Points

Unix. Identify what each of the following unix commands do. If you choose option 4, state what the command does.

a. echo 'this is a test' | tr test asdf
   1. print this is a asdg
   2. print fhid id a fsdf
   3. print ‘this is a test’ and ‘fhid id a fsdf’
   4. None of these

I threw out this question since the tr had two possible replacements for t.for reference, tr uses the last one so the correct answer is “fhid id a fsdf” or 2

b.echo 'this is a test' | sed 's/hi/low/g'
   1. print this is a test
   2. print fhid id a fsdf
   3. print tlows is a test
   4. None of these

3

c. ping amazon.com
   1. print the time taken for a packet to go to brazil and back
   2. print the image of a small yellow duck on the amazon river
   3. just print the ip address of amazon.com
   4. None of these

4. 3 is close but does not mention the most import thing that ping does, namely give the time for a packet to make a roundtrip to amazon.com

d.tar fcz aa.tar.gz
   1. create a compressed tar file of the contents of the current directory
   2. create an uncompressed tar file of the contents of the current directory
   3. Extract all the files from the named tar file
   4. None of these

4. 1 is close, but tar requires that there be files in the file it creates. So, in this case it does nothing

e. alias pwd=‘ls -l’
   1. Execute the command ‘pwd’ when you enter ‘ls -l’
   2. Execute the command ‘ls -l’ when you enter ‘pwd’
   3. Alert the FBI of a name change
   4. None of the above

2

Q6: 10 Points

What is the output of the following program

```
 1  #include<stdio.h>
 2  typedef struct {
 3      int a;
 4  } A;
 5
 6  A func(A fa)
 7  {
 8      printf("    fa %d\n", fa.a);
 9      fa.a=5;
10      printf("    fb %d\n", fa.a);
11      return fa;
12  }
13  int main()
14  {
15      A a;
16      A b;
17      a.a = 1;
18      b = a;
19      b.a = 10;
20      printf("a %d\n", a.a);
21      b = func(a);
22      printf("b %d\n", a.a);
23      printf("c %d\n", b.a);
24  }
```

```
a 1
    fa1
    fb 5
b 1
c 5
```

Generally people did well on this question

Q7: 10 Points

Identify in the following program lines that print exactly the same thing. For instance, lines 19 and 8 will print the same thing. This question does not ask what is printed but what is identical when printed.

```
 1  #include<stdio.h>
 2  typedef struct {
 3      int a;
 4  } A;
 5
 6  A func(A a)
 7  {
 8      printf("%d\n", a.a);
 9      printf("%d\n", &a);
10      a.a=5;
11      printf("%d\n", &a);
12      printf("%d\n", a.a);
13      return a;
14  }
15  int main()
16  {
17      A a;
18      a.a = 1;
19      printf("%d\n", a.a);
20      printf("%d\n", &a);
21      a = func(a);
22      printf("%d\n", &a);
23      printf("%d\n", a.a);
24  }
```

Pairs are:

8,19

9,11 – the address are the same because the object is the same

12,23 – the value is preserved by through the function exit

20,22 – the address of a does not change as a result of assignment. a is statically allocated at compile time. Its address cannot change. When a gets a the value in the assignment on line 21, it sets the fields inside the struct. This is part of the whole “pass by value” thing in C.

Q8: 10 Points

What is the output of this program? Explain?

```
 1  #include<stdio.h>
 2  typedef struct {
 3      int a;
 4  } A;
 5
 6  A func(int i)
 7  {
 8      A a;
 9      a.a=i;
10      return a;
11  }
12
13  int main()
14  {
15      A a[2];
16      A aa = func(1);
17      a[0]=aa;
18      aa=func(2);
19      a[1]=aa;
20      printf("%d\n", a[0].a==a[1].a);
21  }
```

0

The array a[2] is statically allocated and has space for two structs of type A. When aa is assigned to a[0]. that assignment copies the fields inside of aa into a[0], it does not change the pointer address of a[0]. Rather it simply sets the value of a[0].a to 1. Similarly, when aa is assigned to a[1]. All of which goes to say that a[0] and a[1] are totally independent things and remain so regardless of assignments. Hence a[0].a is not equal to a[1].a