1.
Loop when i=0:
Arr[0][0] = 10*0+0 = 0
Arr[0][1] = 10*0+1 = 1
Arr[0][2] = 10*0+2 = 2
Arr[0][3] = 10*0+3 = 3
Arr[0][4] = 10*0+4 = 4
Loop when outer loop i=1:
Arr[1][0] = 10*1+0 = 10
Arr[1][1] = 10*1+1 = 11
Arr[1][2] = 10*1+2 = 12
Arr[1][3] = 10*1+3 = 13
Arr[1][4] = 10*1+4 = 14
Loop when i=2:
Arr[2][0] = 10*2+0 = 20
Arr[2][1] = 10*2+1 = 21
Arr[2][2] = 10*2+2 = 22
Arr[2][3] = 10*2+3 = 23
Arr[2][4] = 10*2+4 = 24
Loop when i=3:
Arr[3][0] = 10*3+0 = 30
Arr[3][1] = 10*3+1 = 31
Arr[3][2] = 10*3+2 = 32
Arr[3][3] = 10*3+3 = 33
Arr[3][4] = 10*3+4 = 34
Loop quits so we got a matrix filled in of
[0 1 2 3 4]
[10 11 12 13 14]
[20 21 22 23 24]
[30 31 32 33 34]
Now printf("%d", *(arr[1]+9));
Now we break down the *(arr[1]+9) into two parts.
*(arr[1]) is basically the pointer to the 2nd row, which is 10. Basically the same thing as *(arr[1][0]).
Now , the pointer position is being added by 9, so we start the pointer at 10, and count to the right by 9. Highlighted in yellow will show the location of the pointer while counting.
[0 1 2 3 4]
[10 11 12 13 14]
[20 21 22 23 24]
[30 31 32 33 34]
So now the new pointer position is at where the value is 24 at at arr[3][4]
So the output will be 24

2.

char *str2= str1;
Means a pointer to a string, is now storing str1
while (*++str1)
Basically goes through the entire string as long as there is a character, and ends if there is no more characters due to how while loops works because while loops works if there is any non-zero value in its parameter, so the pointer being incremented will go through the entirety of the string until the pointer points to nothing.
return (str1 - str2); this is our operation
char *str = "GeorgiaState";
Now GeorgiaState is stored inside the *str pointer which is being passed as the parameter for our method fun.
So basically,. We have *str2 being set as =str1. So now str2 is pointer for the string "GeorgiaState" as well, so currently both *str1 and *str2 = G, and only str1 gets incremented.
Numbering the string GeorgiaState to show array position value for each letter including the null variable because of the fact that it is pointer for string char.
G e o r g i a s t a t e \0
0 1 2 3 4 5 6 7 8 9 10 11 12
So When *str1 = e=1, *str2 =G=0
1-0=1
*str1=o, *str2=G=0
2-0=2
*str1=r=3, *str2=G=0
3-0=3…
Basically, since each value is subtracted by 0, we only count using the value of *str1 so we are just counting the elements in the string including the null value. Georgiastate has 11 letters, but adding the \0 null gives it the 12 character so the output decimal value will print out 12 because it basically counts through the string since it is *str1 - 0, as we enter each array position

3.output:Computers Computers Computers

4.#include <stdio.h>

```
void swap(int *p, int *q) {
   *p = *p + *q;
   *q = *p - *q;
   *p = *p - *q;

int main() {
   int i = 12356, j = 3542030;

   swap(&i, &j);

   printf("swap: i = %d, j = %d\n", i, j);
```

```
    return 0;
}
```

5.
```
#include <stdio.h>

void find_two_largest(int a[], int n, int *largest, int *second_largest) {
    if (n < 2) {
        *largest = *second_largest = -1;
        return;
    }


    if (a[0] > a[1]) {
        *largest = a[0];
        *second_largest = a[1];
    } else {
        *largest = a[1];
        *second_largest = a[0];
    }

    for (int i = 2; i < n; i++) {
        if (a[i] > *largest) {
            *second_largest = *largest;
            *largest = a[i];
        } else if (a[i] > *second_largest) {
            *second_largest = a[i];
        }
    }
}

int main() {
    int a[] = {5, 9, 3, 7, 2, 8};
    int n = sizeof(a) / sizeof(a[0]);
    int largest, second_largest;

    find_two_largest(a, n, &largest, &second_largest);

    printf("Largest: %d\n", largest);
    printf("Second Largest: %d\n", second_largest);

    return 0;
}
```

6.

```
int sum_two_dimensional_array(const int *a, int n) {
    int sum = 0;
    const int *p = a;

    for (int i = 0; i < n * LEN; i++) {
        sum += *p++;
    }

    return sum;
}
```

7. Submitted to dropbox with .c and output file

8.
The scanf will read: i = 12, s= abc34 j=56
So %d reads only the integers and we initialized our integers for 12 but a is not an integer, so it automatically stops reading the a and moves to the next command, which is %s. %s reads the strings, and so it reads the abc and also the 34 because its reading the 34 not as an integer but as a string, so the full string it reads up to is abc34 and only stopped reading because of the space so it executes the next %d for j. For j, it followed the same logic as our i, which only reads up the integers, and since only 56 can be read as an integer and def is a char so it can't read that. The reason why 78 isn't read is because it is separated by the def character and %d reads the first instances of integers, it just ignores anything past def.

9.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

S[] is an array that is storing the string “Hsjodi” and we have a pointer p
P = s calls the first character of the array, so right now *p is pointing to H.
The loop iterates through the entire array, and doing decrementing our *p pointer value. Now it seems that the value being decremented on the pointer is the ASCII values, and we started at the ASCII value of H which is 48. 48-1=47, which is G, so H=G. So doing this for the rest of the array looks like this in Hexadecimals:
H = 48, 48-1=47, 47=G
s = 73, 73-1=72, 72=r
j= 6A, A=10, 6(10) -1 = 69, 69=i
o=6F, 6(15) -1 = 6(14)= 6E, 6E=n
d=64, 64-1 =63, 63=c
i=69, 69-1=68, 68=h
So the final output from the loop is the string Grinch.
Or in short, we can just use the alphabet
10.
the function f takes two character pointers (char *s and char *t) as input, which represents two strings. It returns an integer value that indicates the position of the first character in s that is not found in t.
Using the loops, this function iterates through the characters using pointers stored in s(outer loop) and t(inner loop) and compares each character of s and t. If it finds the same first

character from s that is also found in t, it continues the loop to the next character that matches and continues until it doesn't match and return the index position of that last unmatching character that broke the loop as the solution for p1-s.

(a)3

(b) 0

(c) Returns the index of the first unmatching character or returns the length of the string/array if all are matching.