1. A creates contiguous storage for 100 chars and sets the name x equal to the starting address of that block of memory. B creates a single pointer variable capable of storing the address of a block of characters. In B, you need to set x to point to some actual memory.

2. A compares the addresses stored in two pointers to see if the addresses are equal. B checks if the strings pointed to by p1 and p2 contain the same characters. You would use A if you wanted to see if p1 and p2 pointed to the same place in memory, you would use B when you wanted to see if the strings were the same.

3. return causes control to leave the current function and send a value back to the calling function. exit() causes control to leave the current program and send a value back to the program that called it. Use A when you to leave a function, use B to leave a program.

4. A decrements y and stores the new value in x. B stores in x a value one less than the value in y. In both cases x gets the same value, but only in case A does the value in y change.

5. A reads one white-space-separated string into the array called buffer. B reads one complete line into buffer. Use A to read in words, use B to read in lines.

6. The difference here is strlen() vs sizeof(). Strlen returns the number of characters in a string (not including the null byte) while sizeof() returns the number of bytes used by the variable. In B, the value will be the size of a pointer variable, which is 4 bytes on 32-bit machines and 8 bytes on 64-bit machines.

7. The null byte is not copied to the result array. The simplest fix is to add the line `str2[i] = '\0';` after the loop.

8. Adding 1 to str will pass the address of the second char in str to strlen(). Change the expression to `strlen(str)+1`.

9. You need parentheses around `c = getchar()` or else the comparison to `EOF` will be done before the assignment.

10. a) 'w' b) 'e' c) 'i' d) 0 e) pointer to pointer to char

f)

```
char *tmp;
tmp = s[2];
s[2] = s[3];
s[3] = tmp;
```

```
/* #11A
 * see if list is in order
 */

int in_order( int list[], int len )
{
    int p;

    if ( len>=2 )
      for( p = 0 ; p<len-1 ; p++ )
         if ( list[p] >= list[p+1] )
           return 0;
    return 1;
}


/* #11B
 * add two zero-terminated int arrays
 * put sum in *res
 */
void add_arrays(int *l1, int *l2, int *res )
{
   while( *l1 )
     *res++ = *l1++ + *l2++ ;
   *res = *l1 ;
}

/* #12A */

an array of 25 structs, each struct contains
an int, a ptr, an array of 6 ints, and
an array of 2 ints.
```

```
/* #12B */
/* traverse an array of structs selecting the name
 * and an item from the grade array
 */
void
print_hw_grades(struct student class[], int hw, int len)
{
   int i;

   printf( "HW#%d Name\n", hw );
   for( i=0; i<len ; i++ )
     printf( "%4d %s\n", class[i].grades[hw], class[i].name);
}



/* 13A */

/*
 * traverse a linked list of lists, and for each
 * list in the list, traverse IT looking for a specific
 * value.
 */
void over10()
{
    struct list *listp;
    struct link *linkp;

    for( listp=head.next ; listp ; listp=listp->next )
       for( linkp=listp->words ; linkp ; linkp=linkp->next )
           if ( linkp->value > 10 )
              printf("%s\n", linkp->word );
}
```