1a

```
#include <iostream>
using namespace std;

int main()
   {
       int arr[3] = { 5, 10, 15 };
       int* ptr = arr;

       *ptr = 30;           // set arr[0] to 30


       *(ptr+1) = 20;       // set arr[1] to 20

       ptr += 2;
       ptr[0] = 10;         // set arr[2] to 10

       ptr -=2; //reset position of ptr
       while (ptr <= &arr[2]){
           cout << *ptr << endl;     // print values
           ptr++;
       }
   }
```

1b

The original code does not work because while pToMax was changed in the findMax function, the change was discarded as soon as the function findMax() ended. To preserve the change, pToMax has to be passed to findMax() by reference. Thus replacing **int* pToMax** with **int*& pToMax** fixes the issue.

```
void findMax(int arr[], int n, int*& pToMax){
       if (n <= 0)
           return;       // no items, no maximum!

       pToMax = arr;

       for (int i = 1; i < n; i++)
       {
           if (arr[i] > *pToMax)
                pToMax = arr + i;
       }
   }
```

1c

The program will likely not work because the pointer **ptr** is not pointing to anything in memory. If we initialize a variable (say n) and point **ptr** towards its position in memory, the program will run smoothly.

```
int main()
   {
      int n;
      int* ptr=&n;
      computeCube(5, ptr);
      cout << "Five cubed is " << *ptr << endl;
   }
```

1d

In the while and if conditions the memory addresses are being compared instead of the value at those addresses. We can fix this by adding the * operator to str1 and str2. Also, in the **return** statement, we should be checking that both str1 and str2 have terminated to see whether or not they are equal. We should not be comparing memory addresses like the original code does for the return statement.

```
bool strequal(const char str1[], const char str2[])
   {
      while (*str1 != '\0'  &&  *str2 != '\0')  // zero bytes at ends
      {
          if (*str1 != *str2)  // compare corresponding characters
              return false;
          str1++;             // advance to the next character
          str2++;
      }
      return *str1 == '\0' && *str2=='\0';   // both ended at same time?
   }
```

1e

What the program is doing incorrectly is relying on an array locally initialized inside a function. Memory for this array is not allocated properly so it can be overwritten by an array created by another function like f(), which is undefined behavior and leads to unpredictable program behavior.

2)

```
int main(){


   double* cat; // a
   double mouse[5]; //b
```

```
    cat = &mouse[4]; //c
    *cat = 25; //d
    *(mouse + 3) = 42; //e
    cat = cat-3; //f
    cat[1] = 17; //g
    cat[0] = 54; //h
    bool d = (cat == mouse);//i
    bool b = (*(cat+1)==*cat);//j
}
```

3a)

```
double mean(const double* scores, int numScores){
        double tot = 0;
        int i=0;
        while (i< numScores)
        {
            tot += *(scores+i);
            i++;
            //ptr++;
        }
        return tot/numScores;
}
```

3b)

```
const char* findTheChar(const char* str, char chr){
      for (int k = 0; *str != 0; k++){
          if (*str == chr)
              return *&str;
          str = str+1;
      }
      return nullptr;
}
```

3c)

```
const char* findTheChar(const char* str, char chr){
        while(*str!=0){
            if (*str == chr)
                return *&str;
            str = str+1;
        }

        return nullptr;
```

```
}
```

4)
Prints:
3 //maxwell() compares values at two memory locations passed as arguments. Since 5>4, ptr is //initially initlized to point to array[0]. ptr then has two added to it, shifting it to arr[2]. In the print //statement, we are essentially outputting &array[5]-&array[2] which equals 3
4 //swap2() switches array[0] and array[2]
79 //*(array+1)=79 changes array[1] to 79 from its initial value of 3
-1 //*ptr=-1 set array[0] to -1. swap2() then switched array[0] and array[2] which is why -1 is the //third element of the array printed (index 2)
9 //After ptr was changed to point to arr[2], the ptr[1]=9; statement changed arr[3] (the fourth //element of the array) from 17 to 9
22 //this element was unchanged from the start
19 //this element was unchanged from the start

```
int main(){
   int array[6] = { 5, 3, 4, 17, 22, 19 }; //5 becomes -1, ptr to index 2, 17 becomes
9, 3 becomes 79

   int* ptr = maxwell(array, &array[2]); //This line initializes ptr.
   //maxwell compares the values at the two memory locations and
   //returns the memory location with the higher value.
   //array points to 5 while &array[2] points to 4.
   //Thus, ptr is initialized with the memory location array.

   *ptr = -1; //sets the value at arr (where ptr points) to -1
   ptr += 2; //This moves ptr to the memory location two over. ptr now points to the
value at array[2]
   ptr[1] = 9; //changes value of array[3] to 9 from 17
   *(array+1) = 79; // changes value of array[1] to 79 from 3
   cout << &array[5] - ptr << endl<<endl; //ptr points to &array[2]. Thus, this
statement
   //prints 3 because 5-2=3

   //Array is now {-1, 79, 4, 9, 22, 19} after these changes

   swap1(&array[0], &array[1]); //swap1 doesn't do anything because it changes
pointers inside the function
   swap2(array, &array[2]); //swap2 swaps the values pointed to by the pointers passed
to swap2
   //this swaps -1 and 4 (or in other words, array[0] and array[2])
```

```
   //Array is now {4, 79, -1, 9, 22, 19} after these changes

   for (int i = 0; i < 6; i++)
       cout << array[i] << endl; //this simply prints out all elements in the array

}
```

5)

```
void removeS(char *c){

   char* a = c;
   cout << endl;
   while(*c!='\0'){
       if(!(*c=='S' || *c=='s')){
           *a=*c;
           a++;
       }
       c++;
   }
   *a='\0';
}
```