I.1.

1. Functions “print_usage”, “mymul”, “mydiv”, “myabs”, “myconcat” are defined without declaration

2. switch (type)
   ^
   Cannot pass address to switch statement, only integer or character is supported. Should be changed as
   switch(*type)

3. case "1":
   printf("Multiplication result is %d\n", mymul(a, b));
   ^
   case "2":
   case statements are implemented without break statement. break statement should be added to every case statement.

4. printf("Division result is %d\n", mydiv(a, b));
   ^
   Division between two integers can return float value. So the format parameter should be changed from %d to %f

5. myabs(a);
   ^
   Since, the function does not return anything, it is expected to change the value of 'a' by reference. So reference of 'a' should be passed as argument like
   myabs(&a)

6. int mydiv(int a, int b)
   ^
   Division between two integers can return float value. So the return type should be changed from int to float.

7. return a / b;
   ^
   If b is 0 ,’division by zero’ error will arise. Value of b should be checked before dividing the integers to avoid ‘divide by zero’ error.

8. void myabs(int a)
   ^
   myabs function is called by reference from the main function. So the argument should accept address instead of value. It can be changed as
   void myabs(int *a)

9. a = -a;
   ^

   Since reference is passed as argument instead of value, we have to use pointer to access the value. It can be changed as

   *a = - *a;

10. a = -a;
      ^

    The intention of abs function is to find the absolute value or modulus of integer a. But the logic implemented is as to return negative value of integer a. The logic has to be changed like to check if the integer is less than 0 and then return negative value of that integer.

```
if(a < 0){
    a = -a;
}
```

11. char * str = malloc(strlen(a) + strlen(b));
                  ^

    In C, string must be terminated by '\0'. So there must be space for extra character for string 'str'. It can be changed into

    char * str = malloc(strlen(a) + strlen(b) + 1);

I.2.

From the program, it can be seen that for a particular count, successive elements of src are written to same address pointed by dest. This seems to correct in situations like LED display or socket communication with the microcontroller. In situations like LED display, there will be cases like sending successive elements of data to a same address to blink or display some data in LED.

I.3

a) int a, *b - valid

   a is an integer and b is a pointer to integer.

b) double (fs[])(int) - invalid

   fs is an array of functions returning double accepting int as argument. It is invalid,

Since C cannot have array of functions

c) double *(*f)(double d) - valid

   f is pointer to function accepting single argument double d returning pointer to double.

d) int *const p - valid

p is a constant pointer to integer.

e) int (*cmps[10])(const void *, const void *) - valid

cmps is an array of 10 pointers to function returning integer. The function accepts two arguments of pointer to constant void type(pointing to memory which cannot be changed).

I.4.

a)

```
typedef double (*ARGUMENT[])(int);   // Single Argument typedef

typedef void * (*RETURN)(void);              // Return typedef

RETURN f(ARGUMENT)                    // f() declaration
```

b)

```
void * (*f(double (*[])(int)))(void)        // f declaration without auxillary typedef
```

I.5.

a) Invalid - The compiler will not throw error since void f() prototype means that the function accepts unspecified number of arguments.

b) Valid - The statement is valid for every numeric type since in C, self-comparison always evaluates to true.

c) Valid – The memory for unsigned int is 4 bytes while memory for long is 8 bytes. So there is no chance of arithmetic overflow while casting unsigned int to long.

d) Invalid – It is not fine to malloc() a single character at a time. This will lead to the trouble of handling many pointers. And malloc involves system call, so switching from user space to system space is not efficient and it may affect the performance.

e) Invalid – In C, typedef is a keyword to rename existing datatypes with a new name. It does not define a new type.