You are required to fix all the logical error in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use printf() to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
The function/method ***patternPrint*** accepts an argument *num*, an integer.
The function/method ***patternPrint*** prints *num* lines in the following pattern.

For example, *num* = 4, the pattern should be:

```
1
11
111
1111
```

The function/method ***patternPrint*** compiles successfully but fails to print the desired result for some test cases due to incorrect implementation of the function/method. Your task is to fix the code so that it passes all the test cases.

```c
// You can print the values to stdout for debugging
void patternPrint(int num)
{
    int print=1,i,j;
    for(i=0;i<num;i++)
    {
        for(j=0;j<=i;j++);
        {
            printf("%d ",print);
        }
        printf("\n");
    }
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.
Lisa always forgets her birthday which is on the 5th of July. So, develop a function/method which will be helpful to remember her birthday.

The function/method ***checkBirthDay*** return an integer '1' if it is her birthday else returns 0. The function/method ***checkBirthDay*** accepts two arguments - *month*, a string representing the month of her birthday and *day*, an integer representing the date of her birthday.

The function/method ***checkBirthDay*** compiles successfully but fails to return the desired result for some test cases. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int checkBirthDay(char* month, int day)
3  {
4      if((strcmp(month,"July")) || (day==5))
5              return 1;
6          else
7              return 0;
8  }
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***allExponent*** returns a real number representing the result of exponentiation of base raised to power exponent for all input values. The function/method ***allExponent*** accepts two arguments - *baseValue*, an integer representing the base and *exponentValue*, an integer representing the exponent.

The incomplete code in the function/method ***allExponent*** works only for positive values of the exponent. You must complete the code and make it work for negative values of exponent as well.

Another function/method ***positiveExponent*** uses an efficient way for exponentiation but accepts only positive *exponent* values. You are supposed to use this function/method to complete the code in ***allExponent*** function/method.

Helper Description

The following function is used to represent a positiveExponent and is already implemented in the default code (Do not write this definition again in your code):

```
int positiveExponent(int baseValue, int exponentValue)
{
    /*It calculates the Exponent for the positive value of exponentValue
      This can be called as -
      int res = (float)positiveExponent(baseValue, exponentValue); */
}
```

```
1  // You can print the values to stdout for debugging
2  float allExponent(int baseValue, int exponentValue)
3  {
4      float res =1;
5      if(exponentValue >=0)
6      {
7          res = (float)positiveExponent(baseValue, exponentValue
8      }
9      else
10     {
11         // write your code here for negative value of exponent
12     }
13     return res;
14 }
15
```

Testcase 1:
Input:
5, 2

Expected Return Value:
25.00000

Testcase 2:
Input:
5, -2

Expected Return Value:
0.04000

```
1  // You can print the values to stdout for debugging
2  float allExponent(int baseValue, int exponentValue)
3  {
4     float res =1;
5     if(exponentValue >=0)
6     {
7        res = (float)positiveExponent(baseValue, exponentValue
8     }
9     else
10    {
11       // write your code here for negative value of exponent
12    }
13    return res;
14 }
15
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
You are given a predefined structure Point and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

You must implement the function/method ***isTriangle*** which accepts three points *P1, P2, P3* as inputs and checks whether the given three points form a triangle.

If they form a triangle, the function/method returns an integer 1. Otherwise, it returns an integer 0.

.

Helper Description
The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
typedef struct point
{
  int X;
  int Y;
}Point;
double Point_calculateDistance(Point *point1, Point *point2)
{
```

```
1  // You can print the values to stdout for debugging
2  int isTriangle(Point *P1, Point *P2, Point *P3)
3  {
4      // write your code here
5  }
6
7
```

Testcase 1:
Input:
(3, 4),
(2, 1),
(1, 5)

Expected Return Value:
1

Testcase 2:
Input:
(1, -1),
(0, -1),
(-1, -1)

Expected Return Value:
0

```
1  // You can print the values to stdout for debugging
2  int isTriangle(Point *P1, Point *P2, Point *P3)
3  {
4      // write your code here
5  }
6
7
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***matrixSum*** returns an integer representing the sum of elements of the input matrix. The function/method ***matrixSum*** accepts three arguments - *rows*, an integer representing the number of rows of the input matrix, *columns*, an integer representing the number of columns of the input matrix and *matrix*, a two-dimensional array representing the input matrix.

The function/method ***matrixSum*** compiles unsuccessfully due to syntactical error. Your task is to debug the program so that it passes all test cases.

```
1  // You can print the values to stdout for debugging
2  int matrixSum(int rows, int columns, int **matrix)
3  {
4      int i, j, sum=0;
5      for(i=0;i<rows;i++)
6      {
7        for(j=0;j<columns;j++)
8          sum + = matrix(i)(j);
9      }
10     return sum;
11 }
```

Testcase 1:
Input:

```
3, 3,
[[3, 2, 1],
 [4, 6, 5],
 [7, 8, 9]]
```

Expected Return Value:

```
45
```

Testcase 2:
Input:

```
3, 3,
[[3, 12, 10],
 [14, 61, 51],
 [27, 84, 95]]
```

Expected Return Value:

```
357
```

```
1  // You can print the values to stdout for debugging
2  int matrixSum(int rows, int columns, int **matrix)
3  {
4      int i, j, sum=0;
5      for(i=0;i<rows;i++)
6      {
7        for(j=0;j<columns;j++)
8          sum + = matrix(i)(j);
9      }
10     return sum;
11 }
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***selectionSortArray*** performs an in-place selection sort on the given input list which will be sorted in ascending order.

The function/method ***selectionSortArray*** accepts two arguments - *len*, an integer representing the length of the input list and *arr*, a list of integers representing the input list, respectively.

The function/method ***selectionSortArray*** compiles successfully but fails to get the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Note:

In this particular implementation of selection sort, the smallest element in the list is swapped with the element at first index, the next smallest element is swapped with the element at the next index and so on.

```
1  // You can print the values to stdout for debugging
2  void selectionSortArray(int len, int* arr)
3  {
4      int x=0, y =0;
5      for(x=0; x<len; x++){
6          int index_of_min = x;
7          for(y=x; y<len; y++){
8              if(arr[index_of_min]>arr[x]){
9                  index_of_min = y;
10             }
11         }
12         int temp = arr[x];
13         arr[x] = arr[index_of_min];
14         arr[index_of_min] = temp;
15     }
16 }
17
```

Testcase 1:
Input:
10, [3, 6, 4, 1, 7, 9, 1, 3, 12, 15]

Expected Return Value:
1 1 3 3 4 6 7 9 12 15

Testcase 2:
Input:
9, [3, 3, 3, 3, 3, 3, 3, 3, 3]

Expected Return Value:
3 3 3 3 3 3 3 3 3

```
1  // You can print the values to stdout for debugging
2  void selectionSortArray(int len, int* arr)
3  {
4      int x=0, y=0;
5      for(x=0; x<len; x++){
6          int index_of_min = x;
7          for(y=x; y<len-1; y++){
8              if(arr[index_of_min]>arr[x]){
9                  index_of_min = y;
10             }
11         }
12         int temp = arr[x];
13         arr[x] = arr[index_of_min];
14         arr[index_of_min] = temp;
15     }
16 }
17
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***descendingSortArray*** performs an in-place sort on the given input list which will be sorted in descending order.

The function/method ***descendingSortArray*** accepts two arguments - *len*, an integer representing the length of the input list and *arr*, a list of integers representing the input list, respectively.

5

The function/method ***descendingSortArray*** compiles successfully but fails to get the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void descendingSortArray(int len, int* arr)
3  {
4      int small, pos, i, j, temp;
5      for(i=0; i<=len-1;i++){
6         for(j=i; j<len;j++){
7           temp = 0;
8           if(arr[i]>arr[j]){
9             temp=arr[i];
10            arr[i]=arr[j];
11            arr[j]=temp;
12          }
13        }
14     }
15 }
16
```

You are required to fix all the logical error in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
The function/method ***sameElementCount*** returns an integer representing the number of elements of the input list which are even numbers and equal to the element to its right. For example, if the input list is [4 4 4 1 8 4 1 1 2 2] then the function/method should return the output '3' as it has three similar groups i.e, (4, 4), (4, 4), (2, 2).

The function/method ***sameElementCount*** accepts two arguments - *size*, an integer representing the size of the input list and *inputList*, a list of integers representing the input list.

The function/method compiles successfully but fails to return the desired result for some test cases due to incorrect implementation of the function/method ***sameElementCount***. Your task is to fix the code so that it passes all the test cases.

Note:
In a list, an element at index i is considered to be on the left of index i+1 and to the right of index i-1. The last element of the input list does not have any element next to it which makes it incapable to satisfy the second condition and hence should not be counted.

```
1  // You can print the values to stdout for debugging
2  int sameElementCount(int size, int *inputList)
3  {
4      int i,count =0;
5      for(i=0;i<size-1;i++)
6      {
7          if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
8              count++;
9      }
10     return count;
11 }
12
13
```

Testcase 1:
Input:
11,
[1, 5, 5, 2, 2, 7, 8, 6, 6, 9, 10]

Expected Return Value:
2

Testcase 2:
Input:
5,
[13, 12, 12, 13, 14]

Expected Return Value:
1

```
1  // You can print the values to stdout for debugging
2  int sameElementCount(int size, int *inputList)
3  {
4      int i,count =0;
5      for(i=0;i<size-1;i++)
6      {
7          if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
8              count++;
9      }
10     return count;
11 }
12
13
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***countOccurrence*** return an integer representing the count of occurrences of given value in the input list.

The function/method ***countOccurrence*** accepts three arguments - *len*, an integer representing the size of the input list, *value*, an integer representing the given value and *arr*, a list of integers, representing the input list.

The function/method ***countOccurrence*** compiles successfully but fails to return the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int countOccurrence( int len, int value, int *arr)
3  {
4      int i=0, count = 0;
5      while(i<len){
6          if(arr[i]==value)
7              count += 1;
8      }
9      return count;
10 }
11
```

Testcase 1:
Input:
7, 3, [2, 3, 4, 3, 5, 6, 7]

Expected Return Value:
2

Testcase 2:
Input:
1, 2, [9]

Expected Return Value:
0

```
1  // You can print the values to stdout for debugging
2  int countOccurrence( int len, int value, int *arr)
3  {
4     int i=0, count = 0;
5     while(i<len){
6        if(arr[i]==value)
7           count += 1;
8     }
9     return count;
10 }
11
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***median*** accepts two arguments - *size* and *inputList*, an integer representing the length of a list and a list of integers, respectively.

The function/method ***median*** is supposed to calculate and return an integer representing the median of elements in the input list. However, the function/method **median** works only for odd-length lists because of incomplete code.

You must complete the code to make it work for even-length lists as well. A couple of other functions/methods are available, which you are supposed to use inside the function/method **median** to complete the code.

Helper Description

The following function is used to represent a quick_select and is already implemented in the default code (Do not write this definition again in your code):

```
int quick_select(int* inputList, int start_index, int end_index, int median_order)
{
   /*It calculate the median value
   This can be called as -
   quick_select(inputList, start_index, end_index, median_order)
   where median_order is the half length of the inputList
}
```

```
1  // You can print the values to stdout for debugging
2  float median(int size, int * inputList)
3  {
4      int start_index = 0;
5      int end_index = size-1;
6      float res = -1;
7      if(size%2!=0) // odd size inputList
8      {
9         int median_order = ((size+1)/2);
10        res = (float)quick_select(inputList, start_index, end_
11     }
12     else // even size inputList
13     {
14        // Write code here
15     }
16     return res;
17 }
18
19
```

Testcase 1:
Input:
5,
[2, 40, 23, 52, 37]

Expected Return Value:
37.00

Testcase 2:
Input:
6,
[-24, -16, -8, -4, -54, -1]

Expected Return Value:
-12.00

```
1  // You can print the values to stdout for debugging
2  float median(int size, int * inputList)
3  {
4      int start_index = 0;
5      int end_index = size-1;
6      float res = -1;
7      if(size%2!=0) // odd size inputList
8      {
9         int median_order = ((size+1)/2);
10        res = (float)quick_select(inputList, start_index, end
11     }
12     else // even size inputList
13     {
14        // Write code here
15     }
16     return res;
17 }
18
19
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***manchester*** print space-separated integers with the following property: for each element in the input list *arr*, if the bit arr[i] is the same as *arr*[i-1], then the element of the output list is 0. If they are different, then its 1. For the first bit in the input list, assume its previous bit to be 0. This encoding is stored in a new list.

The function/method ***manchester*** accepts two arguments - *len*, an integer representing the length of the list and *arr* and *arr*, a list of integers, respectively. Each element of *arr* represents a bit - 0 or 1

For example - if *arr* is {0 1 0 0 1 1 1 0}, the function/method should print an list {0 1 1 0 1 0 0 1}.

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void manchester(int len, int* arr)
3  {
4      int* res = (int*)malloc(sizeof(int)*len);
5      res[0] = arr[0];
6      for(int i = 1; i < len; i++){
7        res[i] = (arr[i]==arr[i-1]);
8      }
9      for(int i =0; i<len; i++)
10         printf("%d ",res[i]);
11 }
```

Testcase 1:
Input:
6, [1, 1, 0, 0, 1, 0]

Expected Return Value:
1 0 1 0 1 1

Testcase 2:
Input:
8, [0, 0, 0, 1, 0, 1, 1, 1]

Expected Return Value:
0 0 0 1 1 1 0 0

```c
// You can print the values to stdout for debugging
void manchester(int len, int* arr)
{
    int* res = (int*)malloc(sizeof(int)*len);
    res[0] = arr[0];
    for(int i = 1; i < len; i++){
      res[i] = (arr[i]==arr[i-1]);
    }
    for(int i =0; i<len; i++)
        printf("%d ",res[i]);
}
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use printf() to debug your code.The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
You are given a predefined structure/class ***Point*** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method ***isRightTriangle*** so that it passes all test cases.

Helper Description

The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
typedef struct point
{
  int X;
  int Y;
}Point;
double Point_calculateDistance(Point *point1, Point *point2)
{
```

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
    // write your code here
}

```

Testcase 1:
Input:
(2, -2),
(-2, -2),
(1, -2)

Expected Return Value:
0

Testcase 2:
Input:
(-1, 0),
(2, 0),
(-1, -4)

Expected Return Value:
1

```
1  // You can print the values to stdout for debugging
2  int isRightTriangle(Point *P1, Point *P2, Point *P3)
3  {
4      // write your code here
5  }
6
7
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***multiplyNumber*** returns an integer representing the multiplicative product of the maximum two of three input numbers. The function/method ***multiplyNumber*** accepts three integers- *numA*, *numB* and *numC*, representing the input numbers.

The function/method ***multiplyNumber*** compiles unsuccessfully due to syntactical error. Your task is to debug the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int multiplyNumber(int numA, int numB, int numC)
3  {
4      int result,min,max,mid;
5      max=(numA>numB)?((numA>numC)?numA:numC):((numB>numC)?numB
6      min=(numA<numB)?((numA<numC)?numA:numC):((numB<numC)?numB
7      mid=(numA+numB+numC)-(min+max);
8      result=(max*mid);
9      return result;
10 }
```

Testcase 1:
Input:
5, 7, 4

Expected Return Value:
35

Testcase 2:
Input:
11, 12, 13

Expected Return Value:
156

```
1  // You can print the values to stdout for debugging
2  int multiplyNumber(int numA, int numB, int numC)
3  {
4     int result,min,max,mid;
5     max=(numA>numB)?((numA>numC)?numA:numC):((numB>numC)?numB
6     min=(numA<numB)?((numA<numC)?numA:numC):((numB<numC)?numB
7     mid=(numA+numB+numC)-(min+max);
8     result=(max*mid);
9     return result;
10 }
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.
The function/method ***drawPrintPattern*** accepts *num*, an integer.
The function/method ***drawPrintPattern*** prints the first *num* lines of the pattern shown below.

For example, if *num* = 3, the pattern should be:
11
1111
111111

The function/method ***drawPrintPattern*** compiles successfully but fails to get the desired result for some test cases due to incorrect implementation of the function/method. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void drawPrintPattern(int num)
3  {
4      int i,j,print = 1;
5      for(i=1;i<=num;i++)
6      {
7          for(j=1;j<=2*i;j++)
8          {
9              printf("%d ",print);
10         }
11         printf("\n");
12     }
13 }
14
```

Testcase 1:
Input:
4

Expected Return Value:
1 1
1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1 1 1

Testcase 2:
Input:
1

Expected Return Value:
1 1

```c
// You can print the values to stdout for debugging
void drawPrintPattern(int num)
{
    int i,j,print = 1;
    for(i=1;i<=num;i++)
    {
        for(j=1;j<=2*i;j++)
        {
            printf("%d ",print);
        }
        printf("\n");
    }
}
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***difference_in_dates*** return an integer representing the difference between given two dates. The difference shall always be either a positive number or zero.

The function/method ***difference_in_dates*** accepts two argument - *date1* and *date2*, representing given two Dates.

Developers at ABC Technologies already use a predefined structure ***Date*** containing day, month, and year as members. A collection of functions/methods for performing some common operations on dates is also available. You are supposed to make use of these functions/methods to calculate and return the difference.

```
1  // You can print the values to stdout for debugging
2  int difference_in_dates(Date *date1, Date *date2)
3  {
4      // write your code here
5  }
6
```

Testcase 1:

Input:

```
{day:02, month:05, year:2013},
{day:02, month:06, year:2013}
```

Expected Return Value:

```
31
```

Testcase 2:

Input:

```
{day:01, month:06, year:2011},
{day:01, month:06, year:2012}
```

Expected Return Value:

```
366
```

```
1  // You can print the values to stdout for debugging
2  int difference_in_dates(Date *date1, Date *date2)
3  {
4      // write your code here
5  }
6
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***replaceMinMax*** is supposed to replace all the even elements of the input list with the maximum element of the list, also replace all the odd elements of arr with the minimum element of the list.

The function/method ***replaceMinMax*** accepts two arguments - *size*, an integer representing the size of the input list and *arr*, a list of integers representing the input list.

The function/method ***replaceMinMax*** compiles unsuccessfully due to syntactical error. Your task is to debug the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void replaceMinMax(int size, int* arr)
3  {
4      int i:
5      if(size>0)
6      {
7          int max = arr.0;
8          int min = arr.0;
9          for(i=0;i<size;++i)
10         {
11             if(max<arr[i])
12             {
13                 max = arr[i];
14             }
15             else if(min > arr[i])
16             {
17                 min = arr[i];
18             }
19         }
20         for(i=0;i<size;++i)
21         {
22             if(arr[i] % 2 == 0)
23                 arr[i]=max;
24             else
25                 arr[i]=min;
26         }
27     }
28 }
```

Testcase 1:
Input:
5,
[2, 5, 8, 11, 3]

Expected Return Value:
11 2 11 2 2

Testcase 2:
Input:
9,
[3, 2, 5, 8, 9, 11, 23, 45, 63]

Expected Return Value:
2 63 2 63 2 2 2 2 2

```
// You can print the values to stdout for debugging
void replaceMinMax(int size, int* arr)
{
    int i:
    if(size>0)
    {
        int max = arr.0;
        int min = arr.0;
        for(i=0;i<size;++i)
        {
            if(max<arr[i])
            {
                max = arr[i];
            }
            else if(min > arr[i])
            {
                min = arr[i];
            }
        }
        for(i=0;i<size;++i)
        {
            if(arr[i] % 2 == 0)
                arr[i]=max;
            else
                arr[i]=min;
        }
    }
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***printPattern*** accepts an argument *num*, an integer.

The function/method ***printPattern*** print the first *num* lines of the pattern as shown below.

For example, if *num* = 3, the pattern should be:

```
1 1
2 2 2 2
3 3 3 3 3 3
```

The function/method ***printPattern*** compiles successfully but fails to print the desired result for some test cases. Your task is to debug the code so that it passes all the test cases.

```c
#include<stdio.h>
void printPattern(int num)
{
    int i,j;
    for(i=1;i<=num;i++)
    {
       for(j=i;j<=2*i;j++)
       {
          printf("%d ",j);
       }
       printf("\n");
    }
}
```

Testcase 1:
Input:
4

Expected Return Value:
1 1
2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4 4 4

Testcase 2:
Input:
5

Expected Return Value:
1 1
2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5

```c
#include<stdio.h>
void printPattern(int num)
{
    int i,j;
    for(i=1;i<=num;i++)
    {
       for(j=i;j<=2*i;j++)
       {
          printf("%d ",j);
       }
       printf("\n");
    }
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***printFibonacci*** accepts an integer *num*, representing a number.

The function/method ***printFibonacci*** prints first *num* numbers of the Fibonacci series.

For example, given input 5, the function should print the string "0 1 1 2 3" (without quotes).

The function/method compiles successfully but fails to give the desired result for some test cases. Your task is to debug the code so that it passes all the test cases.

```
1  void printFibonacci(int num)
2  {
3      int i;
4      long sum=0;
5      long num1 = 0;
6      long num2 = 1;
7      for ( i = 1; i < num; ++i)
8      {
9          printf("%ld ", num1);
10         sum = num1 + num2;
11         num2 = sum;
12         num1 = num2;
13     }
14 }
```

Testcase 1:

Input:

23

Expected Return Value:

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 1

Testcase 2:

Input:

14

Expected Return Value:

0 1 1 2 3 5 8 13 21 34 55 89 144 233

```
1  void printFibonacci(int num)
2  {
3      int i;
4      long sum=0;
5      long num1 = 0;
6      long num2 = 1;
7      for ( i = 1; i < num; ++i)
8      {
9          printf("%ld ", num1);
10         sum = num1 + num2;
11         num2 = sum;
12         num1 = num2;
13     }
14 }
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***manchester*** print space-separated integers with the following property: for each element in the input array *arr*, a counter is incremented if the bit *arr*[i] is the same as *arr*[i-1]. Then the increment counter value is added to the output array to store the result.

If the bit *arr*[i] and *arr*[i-1] are different, then 0 is added to the output array. For the first bit in the input array, assume its previous bit to be 0. For example, if *arr* is {0,1,0,0,1,1,1,0}, the function/method should print 1 0 0 2 0 3 4 0.

The function/method ***manchester*** accepts two arguments- *size*, an integer representing the length of the input array; *arr*, a list of integers representing an input array. Each element of *arr* represents a bit, 0 or 1.

The function/method ***manchester*** compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
#include<stdbool.h>
void manchester(int size, int* arr)
{
    bool result;
    int* res = (int*)malloc(sizeof(int)*size);
    int count  =0;
    for(int i = 0; i < size; i++)
    {
        if(i==0)
            result= (arr[i]==0);
        else
            result = (arr[i]==arr[i-1]);
        res[i] = (result)?(0):(++count);
    }
    for(int i=0; i<size; i++)
    {
        printf("%d ",res[i]);
    }
}
```

Testcase 1:

Input:

```
6,
[1, 1, 0, 0, 1, 0]
```

Expected Return Value:

```
0 1 0 2 0 0
```

Testcase 2:

Input:

```
8,
[0, 0, 0, 1, 0, 1, 1, 1]
```

Expected Return Value:

```
1 2 3 0 0 0 4 5
```

```c
#include<stdbool.h>
void manchester(int size, int* arr)
{
    bool result;
    int* res = (int*)malloc(sizeof(int)*size);
    int count  =0;
    for(int i = 0; i < size; i++)
    {
        if(i==0)
            result= (arr[i]==0);
        else
            result = (arr[i]==arr[i-1]);
        res[i] = (result)?(0):(++count);
    }
    for(int i=0; i<size; i++)
    {
        printf("%d ",res[i]);
    }
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***reverseHalfArray*** modify the input list by reversing the input list from the second half.

For example, if the *inputList* is [20, 30, 10, 40, 50], the function/method is expected to modify the *inputList* like [20, 30, 50, 40, 10].

The function/method ***reverseHalfArray*** accepts two arguments - *size*, an integer representing the size of the list and *inputList*, a list of integers representing the given input list, respectively.

The function/method compiles successfully but fails to get the desired result for some test cases. Your task is to debug the code so that it passes all the test cases.

```
1  void reverseHalfArray(int size, int *inputList)
2  {
3      int i, temp;
4      for(i=0; i< size/2 ; i++)
5      {
6          temp = inputList[size-1];
7          inputList[size-1] = inputList[i];
8          inputList[i] = temp;
9          size -= 1;
10     }
11 }
```

Testcase 1:

Input:

```
7,
[1, 2, 3, 4, 5, 6, 7]
```

Expected Return Value:

```
1 2 3 7 6 5 4
```

Testcase 2:

Input:

```
4,
[2, 8, 4, 6]
```

Expected Return Value:

```
2 8 6 4
```

```c
void reverseHalfArray(int size, int *inputList)
{
    int i, temp;
    for(i=0; i< size/2 ; i++)
    {
        temp = inputList[size-1];
        inputList[size-1] = inputList[i];
        inputList[i] = temp;
        size -= 1;
    }
}
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use printf() to debug your code.The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

You are given a predefined structure/class ***Point*** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method ***isRightTriangle*** so that it passes all test cases.

Helper Description

The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
```

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
    // write your code here
}
```

Problem | Test Cases | Output |

Compile and Run

C

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method **printCharacterPattern** accepts an integer *num*. It is supposed to print the first *num* ($0 \le num \le 26$) lines of the pattern as shown below.

For example, if *num* = 4, the pattern is:

a
ab
abc
abcd

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void printCharacterPattern(int num){
3      int i, j;
4      char ch='a';
5      char print;
6      for(i=0;i<num;i++){
7         print = ch;
8         for(j=0;j<=i;j++)
9           printf("%c",ch++);
10        printf("\n");
11     }
12 }
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.
The function/method **countDigits** return an integer representing the remainder when the given number is divided by the number of digits in it.
The function/method**countDigits** accepts an argument - *num*, an integer representing the given number.

The function/method **countDigits** compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int countDigits(int num){
3      int count =0;
4      while( num != 0 ){
5          num = num / 10;
6          count ++;
7      }
8      return ( num % count );
9  }
10
```

Testcase 1:
Input:
782

Expected Return Value:
2

Testcase 2:
Input:
21340

Expected Return Value:
0

```
// You can print the values to stdout for debugging
int countDigits(int num){
    int count =0;
    while( num != 0 ){
        num = num / 10;
        count ++;
    }
    return ( num % count );
}

```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***difference_in_dates*** return an integer representing the difference between given two dates. The difference shall always be either a positive number or zero.

The function/method ***difference_in_dates*** accepts two argument - *date1* and *date2*, representing given two Dates.

Developers at ABC Technologies already use a predefined structure ***Date*** containing day, month, and year as members. A collection of functions/methods for performing some common operations on dates is also available. You are supposed to make use of these functions/methods to calculate and return the difference.

```
1  // You can print the values to stdout for debugging
2  int difference_in_dates(Date *date1, Date *date2)
3  {
4      // write your code here
5  }
6
```

Testcase 1:

Input:

{day:02, month:05, year:2013},
{day:02, month:06, year:2013}

Expected Return Value:

31

Testcase 2:

Input:

{day:01, month:06, year:2011},
{day:01, month:06, year:2012}

Expected Return Value:

366

```
// You can print the values to stdout for debugging
int difference_in_dates(Date *date1, Date *date2)
{
    // write your code here
}

```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***arrayReverse*** modify the input list by reversing its element
The function/method ***arrayReverse*** accepts two arguments - *len*, an integer representing the length of the list and *arr*, list of integers representing the input list, respectively.

For example, if the input list *arr* is {20 30 10 40 50}, the function/method is supposed to print {50 40 10 30 20}.

The function/method ***arrayReverse*** compiles successfully but fails to get the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void arrayReverse(int len, int* arr)
3  {
4      int i, temp, originalLen=len;
5      for(i=0;i<=originalLen/2;i++){
6         temp = arr[len-1];
7         arr[len-1] = arr[i];
8         arr[i] = temp;
9         len -= 1;
10     }
11 }
```

Testcase 1:
Input:
4, [4, 2, 8, 6]

Expected Return Value:
6 8 2 4

Testcase 2:
Input:
3, [11, 20, 17]

Expected Return Value:
17 20 11

```c
// You can print the values to stdout for debugging
void arrayReverse(int len, int* arr)
{
    int i, temp, originalLen=len;
    for(i=0;i<=originalLen/2;i++){
       temp = arr[len-1];
       arr[len-1] = arr[i];
       arr[i] = temp;
       len -= 1;
    }
}
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *print* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***countElement*** return an integer representing the number of elements in the input list which are greater than twice the input number K.
The function/method ***countElement*** accepts three arguments - *size*, an integer representing the size of the input list, *numK*, an integer representing the input number K and *inputList*, a list of integers representing the input list, respectively.

The function/method ***countElement*** compiles unsuccessfully due to syntactical error. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int countElement(int size, int numK, int *inputList)
3  {
4     int i,cou-nt=0;
5     for(i=0,i<size,i++)
6     {
7        if(inputList[i]>2numK)
8
9           cou-nt+=1;
10    }
11    return cou-nt;
12 }
13
14
```

Testcase 1:

Input:

```
7, 3,
[-2, -4, -3, -5, -6, -7, -8]
```

Expected Return Value:

```
0
```

Testcase 2:

Input:

```
6, 13,
[22, 55, 66, 33, 44, 77]
```

Expected Return Value:

```
5
```

```
1  // You can print the values to stdout for debugging
2  int countElement(int size, int numK, int *inputList)
3  {
4     int i,cou-nt=0;
5     for(i=0,i<size,i++)
6     {
7       if(inputList[i]>2numK)
8
9          cou-nt+=1;
10    }
11    return cou-nt;
12 }
13
14
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***removeElement*** prints space separated integers that remains after removing the integer at the given index from the input list.

The function/method ***removeElement*** accepts three arguments - *size*, an integer representing the size of the input list, *indexValue*, an integer representing given index and *inputList*, a list of integers representing the input list.

The function/method ***removeElement*** compiles successfully but fails to print the desired result for some test cases due to incorrect implementation of the function/method ***removeElement***. Your task is to fix the code so that it passes all the test cases.

Note:

```
// You can print the values to stdout for debugging
void removeElement(int size, int indexValue, int *inputList)
{
    int i,j;
    if(indexValue<size)
    {
       for(i=indexValue;i<size-1;i++)
       {
         inputList[i]=inputList[i++];
       }
       for(i=0;i<size-1;i++)
            printf("%d ",inputList[i]);
    }
    else
    {
       for(i=0;i<size;i++)
            printf("%d ",inputList[i]);
    }
}
```

Testcase 1:

Input:

```
9 3
1 2 3 4 5 6 7 8 9
```

Expected Return Value:

```
1 2 3 5 6 7 8 9
```

Testcase 2:

Input:

```
6 6
11 23 12 34 54 32
```

Expected Return Value:

```
11 23 12 34 54 32
```

```c
// You can print the values to stdout for debugging
void removeElement(int size, int indexValue, int *inputList)
{
    int i,j;
    if(indexValue<size)
    {
       for(i=indexValue;i<size-1;i++)
       {
         inputList[i]=inputList[i++];
       }
       for(i=0;i<size-1;i++)
            printf("%d ",inputList[i]);
    }
    else
    {
       for(i=0;i<size;i++)
            printf("%d ",inputList[i]);
    }
}
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***findMaxElement*** return an integer representing the largest element in the given two input lists.
The function/method ***findMaxElement*** accepts four arguments - *len1*, an integer representing the length of the first list, *arr1*, a list of integers representing the first input list, *len2*, an integer representing the length of the second input list and *arr2*, a list of integers representing the second input list, respectively.

Another function/method ***sortArray*** accepts two arguments - *len*, an integer representing the length of the list and *arr*, a list of integers, respectively and return a list sorted ascending order.

Your task is to use the function/method ***sortArray*** to complete the code in ***findMaxElement*** so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int* sortArray(int len, int* arr)
3  {
4      int i=0,j=0,temp=0;
5      for(i=0;i<len;i++)
6      {
7          for(j=i+1;j<len;j++)
8          {
9              if(arr[i]>arr[j])
10             {
11                 temp = arr[i];
12                 arr[i] = arr[j];
13                 arr[j] = temp;
14             }
15         }
16     }
17     return arr;
18 }
19
20 int findMaxElement(int len1, int* arr1, int len2, int* arr2)
21 {
22     // write your code here
23 }
24
```

Testcase 1:

Input:

```
12, [2, 5, 1, 3, 9, 8, 4, 6, 5, 2, 3, 11],
11, [11, 13, 2, 4, 15, 17, 67, 44, 2, 100, 23]
```

Expected Return Value:

```
100
```

Testcase 2:

Input:

```
7, [100, 22, 43, 912, 56, 89, 85]
6, [234, 123, 456, 234, 890, 101]
```

Expected Return Value:

```
912
```

```
1  // You can print the values to stdout for debugging
2  int* sortArray(int len, int* arr)
3  {
4      int i=0,j=0,temp=0;
5      for(i=0;i<len;i++)
6      {
7          for(j=i+1;j<len;j++)
8          {
9              if(arr[i]>arr[j])
10             {
11                 temp = arr[i];
12                 arr[i] = arr[j];
13                 arr[j] = temp;
14             }
15         }
16     }
17     return arr;
18 }
19
20 int findMaxElement(int len1, int* arr1, int len2, int* arr2)
21 {
22     // write your code here
23 }
24
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***manchester*** print space-separated integers with the following property: for each element in the input list *arr*, if the bit arr[i] is the same as *arr*[i-1], then the element of the output list is 0. If they are different, then its 1. For the first bit in the input list, assume its previous bit to be 0. This encoding is stored in a new list.

The function/method ***manchester*** accepts two arguments - *len*, an integer representing the length of the list and *arr* and *arr*, a list of integers, respectively. Each element of *arr* represents a bit - 0 or 1

For example - if *arr* is {0 1 0 0 1 1 1 0}, the function/method should print an list {0 1 1 0 1 0 0 1}.

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void manchester(int len, int* arr)
3  {
4      int* res = (int*)malloc(sizeof(int)*len);
5      res[0] = arr[0];
6      for(int i = 1; i < len; i++){
7        res[i] = (arr[i]==arr[i-1]);
8      }
9      for(int i =0; i<len; i++)
10         printf("%d ",res[i]);
11 }
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***multiplyNumber*** returns an integer representing the multiplicative product of the maximum two of three input numbers. The function/method ***multiplyNumber*** accepts three integers- *numA*, *numB* and *numC*, representing the input numbers.

The function/method ***multiplyNumber*** compiles unsuccessfully due to syntactical error. Your task is to debug the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int multiplyNumber(int numA, int numB, int numC)
3  {
4      int result,min,max,mid;
5      max=(numA>numB)?numA>numC)?numA:numC):(numB>numC)?numB:nu
6      min=(numA<numB)?((numA<numC)?numA:numC):((numB<numC)?numB
7      mid=(numA+numB+numC)-(min+max);
8      result=(max*int mid);
9      return result;
10 }
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***median*** accepts two arguments - *size* and *inputList*, an integer representing the length of a list and a list of integers, respectively.

The function/method ***median*** is supposed to calculate and return an integer representing the median of elements in the input list. However, the function/method **median** works only for odd-length lists because of incomplete code.

You must complete the code to make it work for even-length lists as well. A couple of other functions/methods are available, which you are supposed to use inside the function/method **median** to complete the code.

Helper Description

The following function is used to represent a quick_select and is already implemented in the default code (Do not write this definition again in your code):

```
int quick_select(int* inputList, int start_index, int end_index, int median_order)
{
   /*It calculate the median value
   This can be called as -
   quick_select(inputList, start_index, end_index, median_order)
   where median_order is the half length of the inputList
}
```

```
1  // You can print the values to stdout for debugging
2  float median(int size, int * inputList)
3  {
4      int start_index = 0;
5      int end_index = size-1;
6      float res = -1;
7      if(size%2!=0) // odd size inputList
8      {
9         int median_order = ((size+1)/2);
10        res = (float)quick_select(inputList, start_index, end_
11     }
12     else // even size inputList
13     {
14        // Write code here
15     }
16     return res;
17 }
18
19
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use printf() to debug your code.The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
You are given a predefined structure/class ***Point*** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method ***isRightTriangle*** so that it passes all test cases.
Helper Description
The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
typedef struct point
{
  int X;
  int Y;
}Point;
double Point_calculateDistance(Point *point1, Point *point2)
{
```

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
    // write your code here
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***countOccurrence*** return an integer representing the count of occurrences of given value in the input list.

The function/method ***countOccurrence*** accepts three arguments - *len*, an integer representing the size of the input list, *value*, an integer representing the given value and *arr*, a list of integers, representing the input list.

The function/method ***countOccurrence*** compiles successfully but fails to return the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int countOccurrence( int len, int value, int *arr)
3  {
4     int i=0, count = 0;
5     while(i<len){
6        if(arr[i]==value)
7           count += 1;
8     }
9     return count;
10 }
11
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.
The function/method ***drawPrintPattern*** accepts *num*, an integer.
The function/method ***drawPrintPattern*** prints the first *num* lines of the pattern shown below.

For example, if *num* = 3, the pattern should be:
1 1
1 1 1 1
1 1 1 1 1 1

The function/method ***drawPrintPattern*** compiles successfully but fails to get the desired result for some test cases due to incorrect implementation of the function/method. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void drawPrintPattern(int num)
3  {
4      int i,j,print = 1;
5      for(i=1;i<=num;i++)
6      {
7          for(j=1;j<=2*i;j++);
8          {
9              printf("%d ",print);
10         }
11         printf("\n");
12     }
13 }
14
```

You are required to fix all the logical error in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
The function/method ***sameElementCount*** returns an integer representing the number of elements of the input list which are even numbers and equal to the element to its right. For example, if the input list is [4 4 4 1 8 4 1 1 2 2] then the function/method should return the output '3' as it has three similar groups i.e, (4, 4), (4, 4), (2, 2).

The function/method ***sameElementCount*** accepts two arguments - *size*, an integer representing the size of the input list and *inputList*, a list of integers representing the input list.

The function/method compiles successfully but fails to return the desired result for some test cases due to incorrect implementation of the function/method ***sameElementCount***. Your task is to fix the code so that it passes all the test cases.

Note:
In a list, an element at index i is considered to be on the left of index i+1 and to the right of index i-1. The last element of the input list does not have any element next to it which makes it incapable to satisfy the second condition and hence should not be counted.

```
1  // You can print the values to stdout for debugging
2  int sameElementCount(int size, int *inputList)
3  {
4      int i,count =0;
5      for(i=0;i<size-1;i++)
6      {
7         if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
8            count++;
9      }
10     return count;
11 }
12
13
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.

Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.

The function/method ***median*** accepts two arguments - *size* and *inputList*, an integer representing the length of a list and a list of integers, respectively.

The function/method ***median*** is supposed to calculate and return an integer representing the median of elements in the input list. However, the function/method **median** works only for odd-length lists because of incomplete code.

You must complete the code to make it work for even-length lists as well. A couple of other functions/methods are available, which you are supposed to use inside the function/method **median** to complete the code.

Helper Description

The following function is used to represent a quick_select and is already implemented in the default code (Do not write this definition again in your code):

```
int quick_select(int* inputList, int start_index, int end_index, int median_order)
{
   /*It calculate the median value
   This can be called as -
   quick_select(inputList, start_index, end_index, median_order)
   where median_order is the half length of the inputList
}
```

```
1  // You can print the values to stdout for debugging
2  float median(int size, int * inputList)
3  {
4      int start_index = 0;
5      int end_index = size-1;
6      float res = -1;
7      if(size%2!=0) // odd size inputList
8      {
9         int median_order = ((size+1)/2);
10        res = (float)quick_select(inputList, start_index, end_
11     }
12     else // even size inputList
13     {
14        // Write code here
15     }
16     return res;
17 }
18
19
```

Testcase 1:
Input:
5,
[2, 40, 23, 52, 37]

Expected Return Value:
37.00

Testcase 2:
Input:
6,
[-24, -16, -8, -4, -54, -1]

Expected Return Value:
-12.00

```
1  // You can print the values to stdout for debugging
2  float median(int size, int * inputList)
3  {
4      int start_index = 0;
5      int end_index = size-1;
6      float res = -1;
7      if(size%2!=0) // odd size inputList
8      {
9         int median_order = ((size+1)/2);
10        res = (float)quick_select(inputList, start_index, end
11     }
12     else // even size inputList
13     {
14        // Write code here
15     }
16     return res;
17 }
18
19
```

use printf() to debug your code.The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
You are given a predefined structure/class ***Point*** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method ***isRightTriangle*** so that it passes all test cases.

Helper Description

The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
typedef struct point
{
  int X;
  int Y;
}Point;
double Point_calculateDistance(Point *point1, Point *point2)
{
    /* Return the distance between  point1 and point2;
      This can be called as -
```

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
    // write your code here
}
```

You are required to complete the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use printf() to debug your code.The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
You are given a predefined structure/class ***Point*** and also a collection of related functions/methods that can be used to perform some basic operations on the structure.

The function/method ***isRightTriangle*** returns an integer '1', if the points make a right-angled triangle otherwise return '0'.
The function/method ***isRightTriangle*** accepts three points - *P1, P2, P3* representing the input points.

You are supposed to use the given function to complete the code of the function/method ***isRightTriangle*** so that it passes all test cases.

Helper Description

The following structure is used to represent point and is already implemented in the default code (Do not write these definitions again in your code):

```
struct point;
typedef struct point
{
  int X;
  int Y;
}Point;
double Point_calculateDistance(Point *point1, Point *point2)
{
```

```
// You can print the values to stdout for debugging
int isRightTriangle(Point *P1, Point *P2, Point *P3)
{
    // write your code here
}
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***countOccurrence*** return an integer representing the count of occurrences of given value in the input list.

The function/method ***countOccurrence*** accepts three arguments - *len*, an integer representing the size of the input list, *value*, an integer representing the given value and *arr*, a list of integers, representing the input list.

The function/method ***countOccurrence*** compiles successfully but fails to return the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int countOccurrence( int len, int value, int *arr)
3  {
4      int i=0, count = 0;
5      while(i<len){
6          if(arr[i]==value)
7              count += 1;
8      }
9      return count;
10 }
11
```

Testcase 1:
Input:
7, 3, [2, 3, 4, 3, 5, 6, 7]

Expected Return Value:
2

Testcase 2:
Input:
1, 2, [9]

Expected Return Value:
0

```
1  // You can print the values to stdout for debugging
2  int countOccurrence( int len, int value, int *arr)
3  {
4     int i=0, count = 0;
5     while(i<len){
6        if(arr[i]==value)
7           count += 1;
8     }
9     return count;
10 }
11
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.
The function/method ***drawPrintPattern*** accepts *num*, an integer.
The function/method ***drawPrintPattern*** prints the first *num* lines of the pattern shown below.

For example, if *num* = 3, the pattern should be:
11
1111
111111

The function/method ***drawPrintPattern*** compiles successfully but fails to get the desired result for some test cases due to incorrect implementation of the function/method. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void drawPrintPattern(int num)
3  {
4      int i,j,print = 1;
5      for(i=1;i<=num;i++)
6      {
7          for(j=1;j<=2*i;j++);
8          {
9              printf("%d ",print);
10         }
11         printf("\n");
12     }
13 }
14
```

You are required to fix all the logical error in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to complete the code as in given implementation. We **do not** expect you to modify the approach.
The function/method ***sameElementCount*** returns an integer representing the number of elements of the input list which are even numbers and equal to the element to its right. For example, if the input list is [4 4 4 1 8 4 1 1 2 2] then the function/method should return the output '3' as it has three similar groups i.e, (4, 4), (4, 4), (2, 2).

The function/method ***sameElementCount*** accepts two arguments - *size*, an integer representing the size of the input list and *inputList*, a list of integers representing the input list.

The function/method compiles successfully but fails to return the desired result for some test cases due to incorrect implementation of the function/method ***sameElementCount***. Your task is to fix the code so that it passes all the test cases.

Note:
In a list, an element at index i is considered to be on the left of index i+1 and to the right of index i-1. The last element of the input list does not have any element next to it which makes it incapable to satisfy the second condition and hence should not be counted.

```
1  // You can print the values to stdout for debugging
2  int sameElementCount(int size, int *inputList)
3  {
4      int i,count =0;
5      for(i=0;i<size-1;i++)
6      {
7          if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
8              count++;
9      }
10     return count;
11 }
12
13
```

Testcase 1:
Input:
11,
[1, 5, 5, 2, 2, 7, 8, 6, 6, 9, 10]

Expected Return Value:
2

Testcase 2:
Input:
5,
[13, 12, 12, 13, 14]

Expected Return Value:
1

```
1  // You can print the values to stdout for debugging
2  int sameElementCount(int size, int *inputList)
3  {
4      int i,count =0;
5      for(i=0;i<size-1;i++)
6      {
7          if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
8              count++;
9      }
10     return count;
11 }
12
13
```

You are required to fix all logical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases.

Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***manchester*** print space-separated integers with the following property: for each element in the input list *arr*, if the bit arr[i] is the same as *arr*[i-1], then the element of the output list is 0. If they are different, then its 1. For the first bit in the input list, assume its previous bit to be 0. This encoding is stored in a new list.

The function/method ***manchester*** accepts two arguments - *len*, an integer representing the length of the list and *arr* and *arr*, a list of integers, respectively. Each element of *arr* represents a bit - 0 or 1

For example - if *arr* is {0 1 0 0 1 1 1 0}, the function/method should print an list {0 1 1 0 1 0 0 1}.

The function/method compiles successfully but fails to print the desired result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  void manchester(int len, int* arr)
3  {
4      int* res = (int*)malloc(sizeof(int)*len);
5      res[0] = arr[0];
6      for(int i = 1; i < len; i++){
7        res[i] = (arr[i]==arr[i-1]);
8      }
9      for(int i =0; i<len; i++)
10         printf("%d ",res[i]);
11 }
```

Testcase 1:
Input:
6, [1, 1, 0, 0, 1, 0]

Expected Return Value:
1 0 1 0 1 1

Testcase 2:
Input:
8, [0, 0, 0, 1, 0, 1, 1, 1]

Expected Return Value:
0 0 0 1 1 1 0 0

```c
// You can print the values to stdout for debugging
void manchester(int len, int* arr)
{
    int* res = (int*)malloc(sizeof(int)*len);
    res[0] = arr[0];
    for(int i = 1; i < len; i++){
      res[i] = (arr[i]==arr[i-1]);
    }
    for(int i =0; i<len; i++)
        printf("%d ",res[i]);
}
```

You are required to fix all syntactical errors in the given code. You can click on *Compile & Run* anytime to check the compilation/execution status of the program. You can use *printf()* to debug your code. The submitted code should be logically/syntactically correct and pass all testcases. Do not write the *main()* function as it is not required.
Code Approach: For this question, you will need to correct the given implementation. We **do not** expect you to modify the approach or incorporate any additional library methods.

The function/method ***multiplyNumber*** returns an integer representing the multiplicative product of the maximum two of three input numbers. The function/method ***multiplyNumber*** accepts three integers- *numA*, *numB* and *numC*, representing the input numbers.

The function/method ***multiplyNumber*** compiles unsuccessfully due to syntactical error. Your task is to debug the code so that it passes all the test cases.

```
1  // You can print the values to stdout for debugging
2  int multiplyNumber(int numA, int numB, int numC)
3  {
4     int result,min,max,mid;
5     max=(numA>numB)?numA>numC)?numA:numC):(numB>numC)?numB:nur
6     min=(numA<numB)?((numA<numC)?numA:numC):((numB<numC)?numB
7     mid=(numA+numB+numC)-(min+max);
8     result=(max*int mid);
9     return result;
10 }
```

Testcase 1:
Input:
5, 7, 4

Expected Return Value:
35

Testcase 2:
Input:
11, 12, 13

Expected Return Value:
156

```
1  // You can print the values to stdout for debugging
2  int multiplyNumber(int numA, int numB, int numC)
3  {
4     int result,min,max,mid;
5     max=(numA>numB)?numA>numC)?numA:numC):(numB>numC)?numB:nu
6     min=(numA<numB)?((numA<numC)?numA:numC):((numB<numC)?numB
7     mid=(numA+numB+numC)-(min+max);
8     result=(max*int mid);
9     return result;
10 }
```