Задача 1. Решете задачата на езика C++. Решения на друг език носят нула точки. Отговорите на подточки 1А, 1Б, 1В и 1Г тряба да съвпадат с това, което би извела програмата. При несъответствие отговорът се оценява с нула точки. Решението на подточка 1Д трябва да бъде технически издържано (например не бива да изтича памет, трябва да се спазват добрите практики за структуриране на програмата и т.н.) . Ако решението съдържа сериозни грешки, то се оценява с нула точки.

1А) Какъв ще бъде изходът от изпълнението на следния фрагмент:

```
int calc(int a[5]) {
    int sum = 8;
    for (int i = 0; a[i]; ++i)
        sum += i;
    return sum;
}

int a[7] = {1, 2, 3};
std::cout << calc(a);
```

Отговор: **11**
Масивът a се подава по референция във функцията calc, тъй като int a[5] е еквивалентно на int a[ ], което е еквивалентно на int *a. Във функцията се акумулира сумата sum, с индексите на масива, докато не се достигне индекс, на чиато позиция имаме нулев елемент (което ще се изтълкува като false в предиката за продължаване на for цикъла). Първият нулев елемент е на позиция 3, тъй като подаваме масива {1, 2, 3, 0 , 0, 0, 0} като индексирането започва от 0. Следователно sum = 8 + 0 + 1 + 2 = 11.

1Б) Какъв ще бъде изходът от изпълнението на следния програмен фрагмент?

```
char text[] = "hello", *p = text;
while (*p) std::cout << ++*p++;
```

Отговор: **ifmmp**
Пойнтърът p е от тип char и сочи към първия елемент на масива text. Докато не се достигне терминиращата нула в края на text, буквата към която сочи p ще се инкрементира, ще се принтира на конзолата и след това ще се инкрементира позицията, на която сочи пойнтъра.

1В) Какво ще изведе следният фрагмент (приемаме, че е част от валидна програма):

```
int *pt
int a[3] = {4, 19, 13};
pt = &a[1];
pt += 1;
std::cout << *pt << std::endl;
```

Отговор: **13**
Пойнтърът p сочи към адреса на второто число от масива a. След това инкрементираме този адрес с 1 (той си знае, че е от тип int и знае с колко бита да се премести) и вече сочи към следващия елемент от масива, който е 13.

1Г) В дадените по-долу празни места попълнете какви ще бъдат стойностите на елементите на двата масива A и B след обръщението към функцията f.

```
void f(int *arr1, const int *arr2)
{
    int *p1 = arr1;
    const int *p2 = arr2;
    while (*p2 >= 0) {
        *p1++ = *p2++;
    }
}

int main() {
    int A[4] = {-1, -2, -3, -4};
    int B[4] = {10, 20, 30, -1};
    f(A, B);
}
```

Отговор: **A: [10, 20, 30, -4], B: [10, 20, 30, -1]**
Масивите се предават по референция във фунцкията f, като масива B е подаден с const, което автоматично означава, че функцията няма да променя стойностите му, т.е. масива B се запазва същия какъвто е при подаването. Остана да видим какво се случва с масива A. Назначаваме пойнтър p1, който сочи към първия елемент на масива arr1, който сочи към масива A и аналогично пойнтър p2, който сочи към масива arr2, който пък сочи към масива B (и този пойнтър е const). Докато p2 не посочи елемент от B, който е по-малък от нула, стойността към която сочи p1 ще се актуализира със стойността на която сочи p2 след което и двата поинтъра ще се инкрементират и ще сочат към следващия елемент. Условието за продължение на while цикъла за първи път не е изпълнено, когато p2 сочи към последния елемент на B.

1D) Дадени са структура Point, описваща точка в декартовата координатна система с координати x и y от тип float, и структура Circle, описваща окръжност с център center от тип Point и радиус r от тип float.
Да се дефинира функция findRelativePosition, която определя относителната позиция на две дадени окръжности една спрямо друга. Резултатът от изпълнението на функцията е стойност от изброения тип:

RelativePosition {NO_COMMON_POINTS, TOUCHING, INTERSECTING, SAME} със следния смисъл:

- NO_COMMON_POINTS: без общи точки
- TOUCHING: допиращи се
- INTERSECTING: пресичащи се
- SAME: съвпадат

```
#include <iostream>
#include <cmath>

struct Point {
    float x;
    float y;

    bool operator==(const Point &other) const {
        return x == other.x && y == other.y;
    }

    double getEuclidDist(const Point &other) const {
        return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2));
    }
};

struct Circle {
    Point center;
    float r;
};
enum RelativePosition {
    NO_COMMON_POINTS,
    TOUCHING,
    INTERSECTING,
    SAME
};

RelativePosition findRelativePosition(Circle c1, Circle c2) {
    static const float EPS = 1e-6;
    if (c1.r > c2.r) return findRelativePosition(c2, c1);
    float dist = c1.center.getEuclidDist(c2.center);
    if (dist < EPS) {
        if (fabs(c1.r - c2.r) < EPS) return RelativePosition::SAME;
        return RelativePosition::NO_COMMON_POINTS;
    }
            if (fabs(c1.r + c2.r - dist) < EPS || fabs(c2.r - c1.r - dist) < EPS) return
RelativePosition::TOUCHING;
    if (c1.r + c2.r > dist || c2.r - c1.r < dist) return RelativePosition::NO_COMMON_POINTS;
    return RelativePosition::INTERSECTING;
}

int main() {
    std::cout << findRelativePosition({1, 1, 1}, {1, -2, 2}) << std::endl; // TOUCHING
    std::cout << findRelativePosition({1, 1, 1}, {1.1, 1.1, 3}) << std::endl; // NO_COMMON_POINTS
    std::cout << findRelativePosition({1, 1, 1}, {1.1, 1.1, 0.1}) << std::endl; // NO_COMMON_POINTS
    return 0;
}
```