A. Ticket de Estacionamento

Para resolver este problema basta aplicar as regras dispostas na tabela de tarifas:

```
if (t <= 15) {
    total = 0;
} else if (t > 15 && t <= 60) {
    total = (t - 15) * 0.10;
} else if (t <= 180) {
    total = 45 * 0.10 + (t - 60) * 0.08;
} else if (t <= 420) {
    total = 45 * 0.10 + 120 * 0.08 + (t - 180) * 0.06;
} else {
    total = 45 * 0.10 + 120 * 0.08 + (240) * 0.06 + (t - 420) * 0.02;
}
```

B. Empilhando Caixas

A pilha de caixas pode ser modelada com a estrutura de dados pilha. A cada nova caixa, devem ser feitas duas verificações: se ainda há espaço na pilha, e se o topo tem massa maior ou igual a caixa a ser empilhada.

Esta solução tem complexidade $O(N)$.

C. Tiro ao Alvo

Para resolver este problema, precisamos calcular a distância de cada ponto (x, y) ao centro $(0, 0)$, isto pode ser feito utilizando a fórmula de distância entre dois pontos:

$$d = \sqrt{x^2 + y^2}$$

Com posse da distância d, conseguimos calcular a pontuação do tiro localizado em (x, y).

Repetindo a estratégia para todos os tiros, conseguimos calcular a pontuação final.

D. OCR Bancário

Uma forma de resolver este problema é mapear cada matriz 3×3 que representa um dígito no seu valor inteiro. Isto pode ser feito através de uma estrutura de mapeamento (i.e. `map<string,int>` no C++). Com posse desta estrutura, basta capturar a string correspondente a cada matrix 3×3 e convertê-la no respectivo inteiro através da estrutura de mapeamento.

E. Números Romanos

O problema consiste em duas etapas: converter um número romano para a notação decimal e escrever em romano um número em notação decimal. A primeira rotina pode ser realizada usando um mapa que associe o valor de cada caractere romano ao seu correspondente

decimal:

```
map<char, int> roman { { 'I', 1 }, { 'V', 5 }, { 'X', 10 }, { 'L', 50 },
    { 'C', 100 }, { 'D', 500 }, { 'M', 1000 } };
```

Lido o número romano como uma string, basta identificar, caractere a caractere, o valor decimal correspondente e somar tal valor ao total, se o valor identificado for maior ou igual ao último valor identificado, ou subtrair, caso contrário. Observe que a string deve ser lida de trás para frente.

```
int roman_to_decimal(const string& n)
{
    int dec = 0, last = 0;

    for (auto it = n.rbegin(); it != n.rend(); ++it)
    {
        int value = roman[*it];
        int signal = value < last ? -1 : 1;

        dec += signal * value;
        last = value;
    }

    return dec;
}
```

Agora, para converter de decimal para romano, basta utilizar um algoritmo guloso. Se a tabela de conversão for construída cuidadosamente, não há necessidade de tratar os casos especiais.

```
string decimal_to_roman(int n)
{
    using si = pair<string, int>;
    vector<si> decimal { si("M", 1000), si("CM", 900), si("D", 500),
        si("CD", 400}, si("C", 100}, si("XC", 90), si("L", 50), si("XL", 40),
        si("X", 10), si("IX", 9), si("V", 5), si("IV", 4), si("I", 1) };

    ostringstream os;
    int index = 0;

    while (n > 0)
    {
        string symbol = decimal[index].first;
        int value = decimal[index++].second;

        int d = n / value;

        for (int i = 0; i < d; i++)
```

```
            os « symbol;

        n = n     }

    return os.str();
}
```

F. Transposição Tonal

Para resolver este problema, podemos primeiramente calcular o resultado $x := M \mod 12$. Suponha que o valor de cada nota varie de 0 a 11, sendo 0 para Ab e 11 para G, assim, dado um valor y da nota, sua transposição pode ser calculada como $(y + x) \mod 12$. Com posse do resultado, basta converter o valor numérico para a string representando a nota transposta.

G. Poesia Sucinta

Para resolver este problema, podemos utilizar um mapeamento que, para cada palavra, guarda a ordem em que ela aparece no texto. Assim, quando uma palavra é lida, basta verificar se ela está presente no mapeamento:

- Se ela não estiver, inclua ela associada ao próximo inteiro na ordem sequencial
- Se ela estiver, a substitua pelo inteiro correspondente.

H. Gabaritos

Uma questão simples: os gabaritos são gerados a partir das combinações, com repetições, das alternativas disponíveis. Logo, o valor de N será igual a $20!/(A!B!C!D!E!)$.

Vale atentar ao fato de que uma variável do tipo **`int`** não comporta o valor de 20!, sendo necessário utilizar variáveis do tipo **`long long`**.

I. Genes Cancerígenos

Este é o problema de Casamento de Padrões, um clássico da Ciência da Computação. Devido às restrições de entrada, um algoritmo força-bruta que confronte o padrão contra todas as posições do texto é suficiente. Outros algoritmos como o KMP conseguem resolver este problema em tempo linear no tamanho do texto.

J. Familiares Russos

O problema se torna simples, uma vez removidos os sufixos dos patronímicos e do nome da família. Para isso, basta remover, do fim das respectivas strings, o maior sufixo possível.

```
vector<string> father_suffixes {"ovich", "evich", "ich", "ovna", "evna", "ichna"};
vector<string> family_suffixes {"ev", "ov", "in", "eva", "ova", "ina"};

string
extract_suffix(const string& name, const vector<string>& suffixes)
```

```
{
    auto size = name.size();

    for (auto suffix : suffixes)
    {
        size_t pos = name.rfind(suffix);

        if (pos == size - suffix.size())
            return = name.substr(0, pos);
    }

    return name;
}
```

De posse da rotina de extração de sufixos, basta comparar a string restante para o nome da família para identificar familiares. Se forem familiares, cheque o restante do patronímico para determinar os irmãos.

K. Múltiplos de 3

Embora este seja um problema de matemática, sobre divisibilidade, a implementação em si depende da manipulação de strings, uma vez que a entrada (com até 200.000 dígitos decimais) extrapola a capacidade de variáveis do tipo **`int`** e **`long long`**.

Uma vez lido o valor de N como uma string, para a obtenção dos dígitos d, cada caractere c deve ser convertido, individualmente, para seu valor inteiro correspondente, o que pode ser feito através da subtração do valor `'0'`: **`int`** `d = c - '0'`.

Por fim, para ver se a soma dos dígitos é ou não múltiplo de três, basta usar o operador %: se o resto da divisão da soma por 3 for zero, N será múltiplo de 3.

L. Números de Thabit

A questão diz respeito à classificação de números naturais segundo dois critérios. O primeiro deles é se o número n é ou não número de Thabit. O ponto crítico desta verificação é determinar se o número é ou não uma potência de dois. Há uma manipulação binária que permite verificar tal propriedade em $O(1)$:

```
bool is_power_of_2(int n)
{
    return n and (n & (n - 1)) == 0;
}
```

Na verdade, a operação binária n & $(n-1)$ retorna um número inteiro que resulta do desligamento do bit ligado mais à direita (menos significativo) de n. Se n tiver um único bit ligado (ou seja, for potência de 2), este desligamento resultará em zero. Atente que a primeira condição exclui o zero, que não é potência de 2 mas cuja operação binária também resultaria em zero.

Com esta rotina, verificar se o número é ou não de Thabit pode ser feito da seguinte forma:

```
bool is_Thabit(int n)
{
    if (((n + 1)         return false;

    return is_power_of_2((n + 1)/3);
}
```

A segunda verificação consiste em checar se n é ou não primo. Para isso, pode-se utilizar a rotina abaixo, mais conhecida, que roda em $O(sqrt(n))$:

```
bool is_prime(int n)
{
    if (n < 2)
        return false;

    if (n == 2)
        return true;

    if ((n         return false;

    for (long long i = 3; i * i <= (long long) n; i += 2)
        if (n             return false;

    return true;
}
```