Planejamento de conteúdo para as Oficinas Introdutórias

Clube de Programação da UTFPR

October 5, 2025

Contents

1 Cabeçalhos e Boas Práticas

1.1 bits/stdc++.h

A header file bits/stdc++.h é uma forma simples de incluir quase todas as estruturas de dados mais comuns e funções mais importantes que são utilizadas em programação competitiva.

A header file se encontra em

```
/usr/include/c++/15.2.1/x86_64-pc-linux-gnu/bits/stdc++.h
```

(no meu pc, arch. outras distribuições possuem em outro local esse arquivo).

1.2 namespace std

Utilitário da linguagem C++, é usada como forma de separação de funções e classes. Não é algo que merece ser estudado a fundo para o nosso caso. Utilizamos para simplificar na hora de escrever o código. (e.g. std::cout vs cout)

1.3 int vs long long

Como representamos números inteiros em C++. Devemos cuidar sempre para o tamanho do número que estamos guardando. Existe problemas em que

int é a representação de 32 bits de um inteiro, armazenando números de até, aproximadamente, $2 \cdot 10^9$.

long long é a representação de 64 bits de um inteiro, armazenando números de até, aproximadamente, $4 \cdot 10^18$.

1.4 float vs double

Como representamos números decimais em C++. Para esse caso, por existir problemas de aproximação, sempre preferimos usar o double.

float é a representação em 32 bits.

double é a representação em 64 bits.

1.5 definição de variáveis globais

Em programação competitiva, a definição de variáveis globais pode ser útil. Para declararmos uma variável global, é só declararmos a variável fora dos escopos das funções. O uso das variáveis globais é por gosto, sempre é possível definir variáveis locais e passá-las como parâmetros (ou referências) nas funcões.

1.6 return

return 0 na main é uma boa prática.

1.7 retorno das respostas

Como o programa deve retornar as respostas? usamos o cout para "printar" no terminal as respostas que o problema pede. Devemos nos atentar que o programa deve retornar exatamente da forma que for pedida no problema. Espaços entre números, quebras de linha, palavras, etc. (e.g. Se o problema pedir para printar "SIM", ele não aceitará respostas como "Sim", "sim", ...)

1.8 acelerar leitura por cin

A implementação do cin padrão do C++ pode ser devagar demais para alguns problemas como limites muito apertados. Devido a isso, utilizamos as seguintes linhas para acelerar a leitura O significado por trás do que as linhas fazem não é

algo que necessitamos saber, mas é um comportamento interessante de aprender sobre a linguagem.

```
ios_base::sync_with_stdio(false)
```

O que faz? Dessincroniza os buffers de stream de I/O do C (scanf, printf) e do C++ (cin e cout)

```
cin.tie(NULL)
```

O que faz? Difícil de explicar, mas o resultado disso é que, assim que buffer de entrada (cin) requerir alguma coisa, o buffer de saída (cout) não necessariamente enviou tudo que guardava. Para garantir que o buffer de saída libere tudo, precisamos dar um flush nele.

1.9 Cabeçalho "Padrão"

Por fim, um cabeçalho padrão que podemos recomendar é algo como:

```
#import <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    return 0;
}
```

2 Resolvendo Problemas

2.1 Leitura de variáveis

Podemos ler as variáveis dadas no exercício usando

```
    int a;
    char b;
    string c;
    cin >> a >> b >> c;
```

Para lermos vetores, há duas maneiras, usando vetores estáticos em C, ou usando vectors do C++. Usamos vectors de C++ da mesma forma que estáticos.

Lembrando: não passamos tamanhos variáveis para vetores estáticos de C.

```
    int a[10];
    int a[n]; // ISSO NAO PODE
    vector<int> b = vector<int>(n) // aqui podemos
    for(int i = 0; i < n; i++){
        cin >> a[i]; // ou
        cin >> b[i]; // ou
    }
```

2.2 Casos de teste

Alguns problemas pedem para lermos casos de teste. Normalmente eles tem essa cara:

```
The first line contains one integer t (1<=t<=100)
Each game is described by one number n (1<=n<=100) // Aqui pode variar

Exemplo
4
1
6
3
98
```

Para entendermos o exemplo, podemos ver que o primeiro número indica quantas linhas para resolvermos teremos. Nesse caso, temos 4 linhas (1, 6, 3, 98), esses são os casos que devemos resolver.

Podemos ler essa entrada dessa forma:

```
1 int t;
2 cin >> t;
3 while(t--){ // pense um pouco porque isso funciona
4     int n;
5     cin >> n;
6     // resolver
7 }
```