A biblioteca padrão de entrada e saída do C++ é a *iostream*. Ela deverá ser incluída no início de todo código em C++, porém, para maratonas, o ideal é que se use cabeçalho que inclui todas as bibliotecas padrões do C++ e também do C: *bits/stdc++.h*
Apesar disso, busque saber quais bibliotecas seu programa realmente irá usar.

O comando *using namespace std* é utilizado para que não se repita o *std* em todos os comandos padrões da linguagem.

```
#include <bits/stdc++.h> // inclusão da biblioteca
using namespace std; // sempre incluir este comando em C++

int main()
{
    /* seu código */
    return 0;
}
```

Em maratonas, sempre utilize o formato de código acima (usando o *int* antes do main() e *return 0* ao **final** do código). Isso evitará dor de cabeça e otimizará o tempo de codificação de sua lógica.

Para entrada e saída, temos os comandos *cin* e *cout*, respectivamente.

```
int n;
cin >> n;   // entrada digitada pelo usuário
cout << n << endl; // saída com quebra de linha
```

Perceba que os acumuladores de entrada são diferentes dos acumuladores de saída.
Entrada: >>
Saída: <<
Todos os comandos realizados dentro do *cout* deverão ser precedidos pelo acumulador (<<).

Em competições, na maioria dos casos, é preferível usar os comandos *scanf* e *printf* da linguagem C ao invés dos comandos de C++ de entrada e saída, pois estes são mais lentos.

Exemplos com entrada e saída – cin e cout:

```
cout <<"Hello World!"<< endl;
```

As mensagens da saída sempre deverão estar sempre entre aspas e entre os acumuladores (<<).

```
double n = 1.45, m = 20.16;
cout << fixed << setprecision(1) << n <<" "<< m << endl;
```

Saída:

```
1.4 20.2
```

O comando *setprecision()* pertence à bibioteca *iomanip,* já o *fixed* pertence à biblioteca *iostream.*
O *fixed* serve para atribuir a precisão (que deve ser definida pelo programador) às variáveis seguintes. Perceba que o valor é arredondado para cima (quando > 5) e arredondado para baixo (quando <= 5) automaticamente.

```
int x = 44;
cout << x * 3 << endl;
cout << 200 <<"\n";
```

Você pode sempre utilizar um valor e/ou realizar operações dentro do *cout*.
Para quebra de linha, você pode também utilizar o '\n' da linguagem C. Tanto o '\n' quanto o *endl* possuem a mesma utilidade.

ESTRUTURAS BÁSICAS

Em C++, as estruturas básicas são iguais às da linguagem C, não havendo nenhuma alteração.

Condicional

```
if(x % 2 == 0)
    cout << x <<" eh par"<< endl;
else
    printf("%d eh impar\n", x);
```

Repetição

```
while(x < 20)
{
    printf("Isso se repetira enquanto x for menor que 20.");
}

for(i = 0; i < n; i++)
{
    cout <<"Repetirei isso n vezes!"<< endl;
}

do
{
    cout <<"Faco isso enquanto x eh diferente de 100\n";
} while(x != 100);
```

Vetores e matrizes

```
int vetorInt[50]; // vetor de inteiros de tamanho 50
bool matrizB[150][200]; // matriz de booleanos de tamanho 150x200
float vF[5]; // vetor de float de tamanho 5
```

Funções e procedimentos

```
int fatorial(int n)
{
    int i, fatorial = 1;
    for(i = 1; i <= n; i++)
        fatorial *= i;
    return fatorial;
}

void mostrarMensagem()
{
    printf("Esta eh uma mensagem com printf!\n");
    cout << "Esta eh uma mensagem com o cout!"<< endl;
}
```

Em C++, é possível declarar variáveis locais dentro dos parâmetros do **for**.

```
for(int i = 0; i < n; i++)
{
    /*  seu código  */
}
```

Funções da biblioteca math.h

floor()	-	arredonda para baixo
ceil()	-	arredonda para cima
sqrt()	-	calcula a raiz quadrada
pow(variável, expoente)	-	calcula a potência
sin()	-	seno
cos()	-	cosseno
tan()	-	tangente
log()	-	logaritmo natural
log10()	-	logaritmo na base 10

Exemplos:

```
double n = 65.881;
cout << fixed << setprecision(2) << floor(n) << endl;
printf("%.2lf\n", ceil(n));
int x = sqrt(36), y = pow(2, 7), s = sin(0), c = cos(0), t = tan(0);
printf("%d\n", x);
cout << y << endl;
cout << s << endl;
printf("%d\n", c);
cout << t << endl;
double ln = 2.718282, l = 10;
cout << log(ln) << endl;
cout << log10(l) << endl;
```

O programa acima vai gerar a seguinte saída:

```
65.00
66.00
6
128
0
1
0
1.00
1.00
```

QUESTÕES DO URI PARA RESOLVER

Entre no seguinte link: URI – Iniciante
Resolva o máximo de problemas que conseguir nesta seção Iniciante. Isso lhe permitirá entender o funcionamento das questões e como as saídas são cobradas.

STRINGS

Strings são cadeias de caracteres. Em C++, estão presentes na biblioteca *string*. Diferente dos vetores de caracteres em C, as strings em C++ não precisam ter um tamanho definido na declaração, pois são dinâmicas.

Entrada e saída

```
string S;
cin >> S;
cout << S << endl;
```

Comparação

```
string s = "Lol > Dota";
string t = "Dota > Lol";

if(s == t)
    cout <<"As strings sao iguais."<< endl;
else
    cout <<"As strings sao diferentes."<< endl;
```

Comandos

```
string S = "Carreta Furacao";

int T = S.size(); // inteiro T pega o tamanho da string S
cout <<"Tamanho da string: "<< T << endl;

// a string " Hurricane Cart" é inserida na string S a partir da 7ª posição
S.insert(7, " Hurricane Cart");
cout <<"Apos a insercao: "<< S << endl;

// a string será apagada da posição 0 até a posição 23
S.erase(0, 23);
cout <<"Apos a remocao: "<< S << endl;
```

Saída:

```
Tamanho da string: 15
Apos a insercao: Carreta Hurricane Cart Furacao
Apos a remocao: Furacao
```

É possível também criar um vetor de strings.

```
string vetorDeString[3]; // vetor de string de 3 posições
vetorDeString[0] = "This ";
vetorDeString[1] = "is ";
vetorDeString[2] = "an array.";
for(int i = 0; i < 3; i++)
{
    cout << vetorDeString[i]; // mostra o vetor de string na posição i
}
printf("\n"); // quebra de linha
```

Saída:

```
This is an array.
```

Dicas iniciantes:

Quando o problema solicitar 'vários' casos de teste, sem especificar a quantidade, é recomendável usar o EOF. Exemplo:

```
while(scanf("%d", &x) != EOF) // entrada pelo usuário
{
    /*  seu código  */
}
```

No exemplo acima, a condição de parada é o fim de arquivo (*end of file – EOF)*, porém, neste caso, como o programa não está lendo nenhum arquivo em específico, a quantidade de casos de teste é 'infinita'. Nas competições, esse modo de leitura é muito útil quando não se especifica a quantidade.

Quando o problema especifica a quantidade de casos de teste, é possível fazer de duas formas:

```
scanf("%d", &n);
while(n--) // variável decrementa a cada iteração
{
    /*  seu código  */
}

scanf("%d", &n);
for(int i = 0; i < n; i++) // variável não é alterada
{
    /*  seu código  */
}
```

Perceba que a variável de casos de teste, digitada pelo usuário, não é alterada como no primeiro exemplo. Utilize cada uma de acordo com as necessidades do problema, isto é, se não for utilizar a variável de casos de teste após digitá-la, faça como no primeiro exemplo. Desse modo, o tempo de codificação será otimizado.