Lista 1 – Programas básicos em C++

Introdução

Esse exercício tem por objetivos fazer você praticar sua habilidade de **interpretar especificações de problemas**, aliada a sua capacidade de **projetar** e **implementar algoritmos**.

Além disso, o desenvolvimento desse trabalho oferecer uma oportunidade prática para utilizar elementos básicos de programação em C++, como laços, condicionais simples e composto, tipos de dados básicos e heterogêneos, expressões lógicas, passagem de parâmetros, criação de funções, leitura de escrita de informações a partir da entrada e saída padrão, dentre outros.

Adicionalmente, você deverá ter contato com alguns elementos da STL (Standard Template Library) do C++, mais especificamente o uso de vetor dinâmico e estático, bem como alguns algoritmos básicos de manipulação de dados, dependendo das necessidades dos algoritmos que você projetar.

1 - Negativos 5

Escreva um programa em C++ que recebe 5 valores inteiros da entrada padrão, conta quantos destes valores são negativos e imprime esta informação na saída padrão. Veja abaixo exemplo de entrada e sada esperada.

Exemplos de Entrada/Saída

Entrada	Saída
`-1 -2 0 1 2`	`2`
`1 2 3 4 5`	`0`

Conchecimentos Necessários

Leitura e escrita da entrada padrão, laços e condicionais

2 - Intervalos

Escreva um programa em C++ que lê um número não conhecido de valores inteiros e conta quantos deles estão em cada um dos intervalos [0; 25), [25; 50), [50; 75), [75; 100) e fora desses intervalos. Para ler um número indeterminado de valores basta incluir o comando de extração associado ao `std::cin` como condição de parada em um laço (ver abaixo).

```
int x;
while( cin >> std::ws >> x) {
```

```
   // realização da contagem em relação aos intervalos
}
```

Após encerrada a entrada de dados, o programa deve imprimir a **porcentagem** de números para cada um dos quatro intervalos e de números fora deles, nessa ordem, um por linha e representados com quatro casas de precisão. Para definir a precisão use a função `std::setprecision(4)` antes do uso do `cout`.

Exemplos de Entrada/Saída

Entrada	Saída
-9 -8 1 5 15 20 25 30 35 40 45 46 47 50 55 60 78 85 90 99 100 120	18.18 31.82 13.64 18.18 18.18
25 30 35 40 45 46 47 78 85 90 99 100 120	0 53.85 0 30.77 15.38

Conhecimentos Necessários

Leitura de entrada padrão, escrita em saída padrão, laços, condicionais, type casting, saída formatada, uso de vetores.

3 - Soma Vizinhos

Escreva um programa em C++ que lê um numero não determinado de pares de valores inteiros, m e n, onde -10000 ≤ n ≤ 1000, e para cada par calula e escreve a soma dos n primeiros inteiros consecutivos a partir de m (inclusive), se n > 0; ou a soma dos n primeiros inteiros antecedentes a partir de m (inclusive), se n < 0. Por exemplo, se a entrada for `1 5` o programa deve imprimir na sada padrão o resultado de `1 + 2 + 3 + 4 + 5`, ou seja, `15`. Por outro lado, se a entrada for `1 -5` o programa deve imprimir na sada padrão o resultado de `1 + 0 + (-1) + (-2) + (-3)`, ou seja, -5.

Exemplos de Entrada/Saída

Entrada	Saída
`4 6`	`39`
`10 -4`	`34`
`39 0`	`39`

Conhecimentos necessários

Leitura de entrada padrão, escrita em saída padrão, laços e condicionais.

4 - Fibonacci

Implemente uma função em C++ chamada fib_below_n que recebe um valor inteiro positivo n e armazena os termos da série de Fibonacci inferiores a n em um vetor do tipo `std::vector`. A função deve então retornar esse objeto usando o comando return. A classe `std::vector` representa um tipo de container que faz parte da biblioteca padrão/STL(c++ tem um conjunto de bibliotecas padrão que provêem várias estruturas úteis inexistentes em C) e representa a estrutura de dados lista dinâmica.
A função deve ter a seguinte assinatura:

```
std::vector<int> fib_below_n ( unsigned int n );
```

Relemebrando: A sequência de Fibonacci define-se como uma sequencia de números inteiros onde os dois primeiros termos são iguais a 1 e cada termo seguinte é a soma dos dois termos imediatamente anteriores. Desta forma se fosse fornecido ao programa uma entrada n = 15 o mesmo deveria produzir a seguinte sequência de termos da série: {1; 1; 2; 3; 5; 8; 13}.

Conhecimentos Necessários

Utilização de funcões, retorno de função, utilização da classe `std::vector` da STL, laços

5 - Minmax

Escreva uma função em C++ chamada min_max que recebe como parâmetro um vetor V com n números inteiros, identifica e retorna um par de valores correspondentes aos índices da primeira ocorrência do menor elemento e da última ocorrência do maior elemento presente em V. Por exemplo, se a entrada fosse `V={5,4,1,3,1,10,7,10,6,8}` a função deveria retornar o par `{2,7}`, correspondente às posições do primeiro '1' em V[2] e do último '10' na posição V[7].

Um par de valores pode ser retornado por uma função através de um struct, vetor ou utilizando a classe utilitária `std::pair`, nesta questão usaremos a `std::pair`. Assim a função deve ter a seguinte assinatura:

```
std::pair <size_t ,size_t> min_max ( int V[], size_t n );
```

Conhecimentos necessários

Utilização de função, passagem de vetor por parâmetro, utilização da classe `std::pair`, retorno de funçao com tipo composto, condicionais, laços.

6 - Inverter

Escreva uma função em C++ chamada reverse que recebe como parâmetro uma referência para um vetor estático de strings, implementado com `std::array` , e inverte a ordem dos seus elementos da forma mais eficiente possível.
Por exemplo, considere o vetor A contém as seguintes strings: ["um", "dois", "três", "quatro"], após a execução da função o vetor A deverá ter seus elementos na seguinte ordem: ["quatro", "três", "dois", "um"].

A classe `std::array` representa um vetor estático de memória contígua que faz parte da biblioteca padrão. A função reverse deve ter a seguinte assinatura:

```
template <size_t SIZE>
void reverse ( std::array <std::string, SIZE> &arr );
```

Note nesta assinatura a presenta da palavra chave **template**. Essa construção permite que a variável SIZE (inteiro longo sem sinal) seja definida em tempo de compilação por quem invocar sua função. Uma das vantagens de usar uma classe para definir o vetor, ao invés de usar um vetor tradicional do C++, é que a classe `std::array` possui várias funcionalidades já implementadas e disponíveis para uso. Por exemplo, se você desejar recuperar a quantidade de elementos em arr basta invocar o método size() , como em: `size_t tamanho = arr . size ();`, com isso não precisamos passaro comprimento de arr para a função.
Outro ponto a se observar é uso do operador de referncia **&**, associado ao parâmetro arr. Essa construção faz com que a função receba uma referencia para a variável que é passada no parâmetro quando a função é usada, fazendo com que operações em arr sejam refletidas fora do escopo da função quando ela acaba. De forma similar ao que ocorria com vetores quando passados para funções em C, por isso a função é void, não precisando retornar o vetor na ordem reversa.

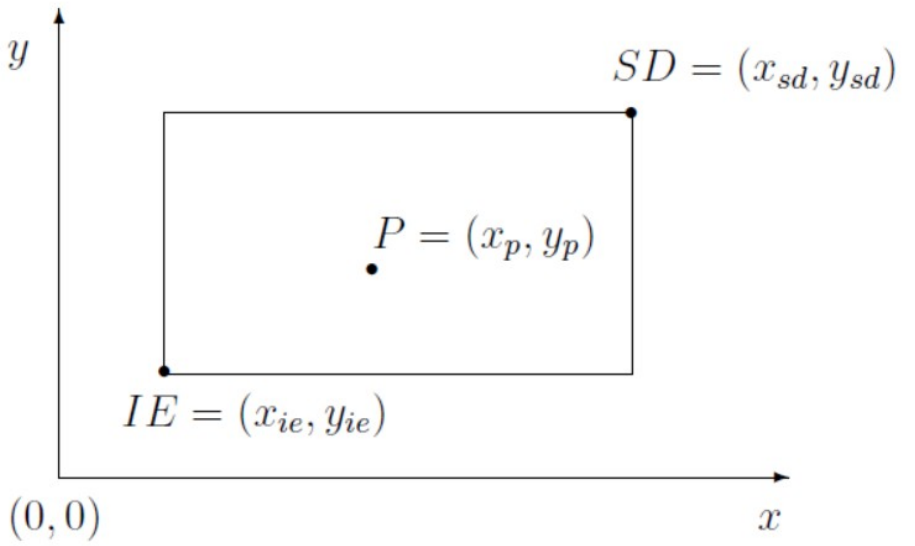
Da mesma forma que C, quando funções em c++ recebem vetores básicos (conhecidos como raw arrays, ou aqueles que usam os []) a função sempre recebe uma referencia, implicando que mudanças nos vetore são refletidas fora do escopo da função. No entanto, nesta função usamos o objeto do tipo `std::array`, neste caso, assim como quando usamos `std::vector` ou qualquer outro objeto, por padrão a **passagem é por valor**, implicando que mudanças no vetor recebido não são refletidas quando a função acaba. Para mudar esse comportamento usamos o **&** na frente do parâmetro desejado, fazendo com que a **passagem seja por referência**, assim o objeto `std::array` nesta função se comporta de forma similar aos vetores comuns de C.

Conhecimentos Necessários

Utilização de funções, strings, passagem de parâmetro por referência, uso da classe `std::array`, laços, uso da função `std::swap`.
Observação: Não é necessário criar um outro vetor para inverter os valores presentes no vetor passado por referência. A inversão pode ser feita internamente, ou seja, dentro do próprio vetor, com a ajuda de variáveis auxiliares.

7 - Ponto em retângulo

Considerando a estrutura abaixo para representar as coordenadas Cartesianas de um ponto no plano bidimensional (2D), implemente uma função em C++ que verifica se um ponto P = (x_p,y_p), determinado por suas coordenadas cartesianas, está localizado **dentro, na borda ou fora** de um retângulo definido por dois pontos: o canto inferior esquerdo IE = (x_{ie},y_{ie}) e o canto superior direito SD = (x_{sd}, y_{sd}).



Struct que descreve um ponto:

```
struct Ponto {
  int x; // ! < coordenada X do ponto.
  int y; // ! < coordenada y do ponto.
  // Construtor padrão
  Ponto ( int xi =0, int yi =0 ) : x{xi}, y{yi} // copia args para os campos x e y
  { /* nada */ }
};
```

A função a ser criada deve receber 3 pontos como referências constante representando, respectivamente, os dois pontos, IE e SD, que definem um retângulo, e o ponto a ser testado P. **Assuma** que os pontos IE e SD definem um retângulo válido, i.e IE≠SD (pelo menos uma coordenada de um ponto é diferente do outro). Dessa forma a função deve ter a seguinte assinatura:

```
int pt_in_rect ( const Ponto &R1, cons t Ponto &R2, const Ponto &P );
```

A seguir, a função deve realizar testes e indicar se o ponto P está dentro, na borda ou fora do retângulo, retornando, respectivamente, os valores 0, 1 ou 2 para cada caso ora descrito.

Se desejar tornar seu código mais inteligível, você pode utilizar a enumeração abaixo:

```
enum location_t : int { INSIDE =0, BORDER =1, OUTSIDE =2 };
```

Neste caso, modifique na assinatura da função o tipo do retorno para que seja `location_t`.

Conhecimentos necessários

Utilização de funções, tipos heterogêneos (struct), passagem de parâmetro por referência constante, uso de enumerações, condicionais, expressões lógicas.

Observações: Veja que, diferente das definições em C, definimos uma função dentro da struct. Esta função, especialmente, permite inicializar a struct de uma forma mais simples, ao invés de setar os valores de x e y ou usar o inicializador inline de C. Para instanciar uma variável do tipo da struct Ponto e colocar valores específicos de x e y apenas fazemos: `Ponto p1 = Ponto(12,32);`. Essa forma é relativamente simples, e talvez fosse melhor usar a inicialização em C (`Ponto p1 = {12,32}`), mas usando os construtores, podemos fazer coisas mais complexas, como chamar funções ou realizar checagens durante a incialização dos campos.

8 - Ponto em Retângulo 2

Implemente um programa em C++ que recebe da entrada padrão um número indeterminado de linhas, cada uma correspondendo a um caso de teste. Cada caso de teste contém informções correspondentes a um possível retângulo e um ponto, ambos definidos no plano Cartesiano 2D. Para cada caso de teste o programa deve (1) verificar se o retângulo é válido e, em caso verdadeiro (2) classificar o ponto em relação ao retângulo em uma das três possibilidades: fora, na fronteira, ou dentro do retângulo.

Um retângulo é considerado válido se e somente se pelo menos uma das quatro coordenadas dos vértices que o define for diferente, ou seja R1≠R2. Portanto, o programa deve aceitar os chamados retângulos "degenerados" que formam uma linha vertical ou horizontal, como por exemplo: IE(2; 5) x SD(2; 7) ou IE(-53;-4) x SD(-5;-4).

A classificação do posicionamento do ponto em relação ao retângulo deve ser feita através da invocação da função implementada na Ponto em Retângulo. Lembre que para a função pt_in_rect funcionar corretamente é necessário informar como argumentos as coordenadas do canto inferior esquerdo e superior direito. Portanto, seu programa deve analisar os vértices de entrada e fazer os ajustes necessários (por exemplo, criando novos pontos a partir das coordenadas originais) para satisfazer o pré-requisito da função de classificação.

Cada linha lida da entrada padrão deve corresponder a um caso de teste, tendo o formato: x_1 y_1 x_2 y_2 x_3 y_3, onde $-1000 \leq x_i, y_i \leq 1000$. Os quatro primeiros valores representam as coordenadas de dois vértices quaisquer do retângulo: R1 = (x_1, y_1) e R2 = (x_2, y_2). Note que esses dois vértices podem representar qualquer um dos quatro possíveis cantos de um retângulo: inferior esquerdo, inferior direito, superior esquerdo, ou superior direito. Os últimos dois valores representam as coordenadas do ponto, P = (x_3, y_3), a ser testado contra o retângulo definido na mesma linha.

Exemplo de entrada/saída

Entrada	Saída
`-3 -1 3 1 0 0`	inside
`2 2 9 7 4 2`	border
`7 9 2 2 2 7`	border
`4 5 4 5 -1 -2`	invalid
`-3 0 5 5 1 6`	outside
`-3 0 5 5 6 4`	outside
`1 2 -5 7 -1 6`	inside
`3 7 -2 -2 2 2`	inside

Conhecimentos necessários

Utilização de funções, reutilização de código, tipos heterogêneos (struct), passagem de parâmetro por referência constante, enumerações, Geometria 2D, condicionais, expressões lógicas.