Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Linguagem de Programação I • DIM0120

◁ 1ª Avaliação, 17 de abril de 2018 ▷

1. [3.0 pts] Considere um algoritmo de ordenação baseado no *bubble sort* no qual a direção de borbulhamento é alternada a cada iteração. Suponha, por exemplo, que desejamos ordenar um arranjo em ordem não decrescente: em uma iteração, o menor elemento "borbulha" para cima, na direção do início do arranjo; na próxima iteração, o maior elemento "borbulha" para baixo, na direção do final do arranjo; e assim por diante até o vetor estar ordenado. Escreva uma função em C++ que implementa esta ideia com ponteiro `void`, passando também a função de comparação como um ponteiro. Escreva também um programa `test_sort.cpp` para testar sua função. Uma possível assinatura para esta função seria:

```
using Compare = int( const void *, const void * );
void sort_bubble_2( const void *first, const void *last, size_t size,
                    Compare *cmp );
```

- `first`, `last` - ponteiros que definem o intervalo de elementos para ordenar.
- `size` - tamanho de um elemento do intervalo em bytes.
- `cmp` - função de comparação que retorna -1 se $a < b$, 0 se $a = b$ e 1 se $a > b$. A assinatura da função de comparação deve ser equivalente a

  ```
  int cmp( const void * a, const void *b );
  ```

2. [2.0 pts] Desenvolva uma função `array_of_selected` que *(i)* recebe como parâmetros 2 ponteiros para inteiros, [`first`, `last`), que definem um intervalo de números inteiros, e um ponteiro para uma função predicado `p`, *(ii)* copia para um novo vetor alocado dinamicamente todos os elementos de [`first`, `last`) que satisfazem o predicado `p` e *(iii)* retorna um `std::pair<int*, int*>` contendo 2 ponteiros para inteiros que definem um intervalo [`a_first`, `a_last`) correspondente ao novo vetor criado pela função. O intervalo [`first`, `last`) não deve ser alterado pela função e o tamanho do novo vetor criado deve ser corresponder *exatamente* ao número de elementos do intervalo [`first`, `last`) que satisfazem o predicado `p`. O protótipo da função pode ser:

```
using UnaryPredicate = bool ( const int * );
std::pair<int*, int*> array_selected( const int *first, const int *last,
                                      UnaryPredicate *p );
```

- `first`, `last` - ponteiros que definem o intervalo de elementos para examinar.
- `p` - predicado unário que retorna `true` para o elemento desejado. A assinatura da função predicado deve ser equivalente a

  ```
  bool pred( const int * a );
  ```

Obs. Para retornar um par com dois ponteiros, digamos `first` e `last`, basta usar:
`return std::make_pair( first, last );`

3. [5.0 pts] Um veículo terrestre não tripulado (VTNT) precisa atravessar uma cadeia de montanhas e vales, o que inclui várias subidas e descidas. Quando o VTNT sobe, ele utiliza a carga da sua bateria para ativar o motor; quando desce, ele recupera a energia que é armazenada na bateria. O processo de recarga da bateria é ideal: na descida, cada Joule de energia potencial gravitacional é convertido para um Joule de energia elétrica a ser armazenado na bateria. A bateria tem uma capacidade limitada de armazenamento e, portanto, quando atingir tal capacidade, a energia (excedente) gerada na descida é perdida.

Escreva um programa em C++ `vtnt.cpp` que lê, a partir de um arquivo de entrada fornecido por linha de comando, uma sequência de $m > 0$ linhas, onde cada linha contém $n > 0$ inteiros (positivos ou negativos), separados por 1 ou mais espaços em branco, correspondente às altitudes que deverão ser transpostas pelo VTNT. Cada linha do arquivo de entrada corresponde a uma instância do problema e deve gerar uma linha de saída em um arquivo de resposta, `saida.txt`, contendo *apenas* um único inteiro correspondente a *capacidade mínima de carga que a bateria deve ter para completar a jornada de ida e volta ao ponto de partida*. Assuma que o arquivo de entrada só possui entradas válidas. O VTNT inicia a jornada com a bateria totalmente carregada. Veja na Figura 1 um exemplo de entrada válida.

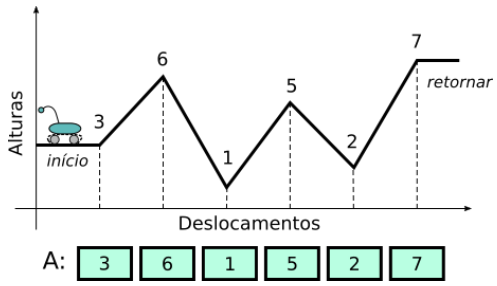

Figura 1: Possível entrada de dados e sua interpretação visual. A carga mínima, neste caso, seria de 6 unidades.

Exemplo de entrada

```
3 8 1 5 2 7
2   3 1 6
2
5  -1 2 6
3 8  2 5 1 7
3 2  5 1  8 7
3 8 -2 5 1  7 -5 10 2 0
1 3 2  4
4 9  2 5 1 7 1 5 2 9 4
-1 -2 -3 10 0 -2 5 -15
3 6 1 5 2 7
```

Exemplo de saída

```
7
5
0
7
7
7
15
3
8
25
6
```

∽ FIM ∽