Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0030 – Linguagem de Programação I

Docente: Umberto S. Costa

Problema: desenvolvimento de habilidades de programação na linguagem C++.

Subproblema 3: alocação dinâmica, gerenciamento de memória, arquivos.

Produto do subproblema: (*i*) resumo das principais características e recursos C++ identificados durante a exploração das questões deste subproblema (até duas páginas, podendo haver apêndices); (*ii*) respostas às questões abaixo; e (*iii*) código-fonte dos programas implementados.

Data de entrega via SIGAA: 29 de agosto de 2017.

Instruções: neste problema o aluno deve consultar as referências indicadas pelo docente para se familiarizar com os recursos necessários à criação de programas simples em C++, sem prejuízo à consulta de outras fontes como manuais e tutoriais. Usar as questões e programas mostrados a seguir como guia para as discussões em grupo e para orientar a exploração da linguagem C++. Para facilitar o aprendizado, recomenda-se que o aluno compare os recursos e conceitos de C++ com seu conhecimento prévio acerca de outras linguagens de programação. Leia e modifique os códigos mostrados e utilize os conceitos e recursos explorados para a criar os programas solicitados. Recursos exclusivos da linguagem C devem ser ignorados e substituídos por seus correspondentes em C++.

Questões[1]:

1. Considere o programa a seguir:

```
#include <iostream>

int main(int argc, char *argv[])
{
  int* pointer{new int{42}};
  std::cout << *pointer << std::endl;

  *pointer = 10;
  std::cout << *pointer << std::endl;
```

[1]Em parte inspiradas em *Exploring C++ 11*, Ray Lischner. Alguns programas foram retirados desta mesma fonte.

```
  delete pointer;                                                    10
  pointer = nullptr;                                                 11
                                                                     12
  //std::cout << *pointer << std::endl;                              13
                                                                     14
  long* good{new long{}};                                            15
  std::cout << *good << std::endl;                                   16
  delete good;                                                       17
                                                                     18
  // delete good;                                                    19
                                                                     20
  good = nullptr;                                                    21
                                                                     22
  // delete good;                                                    23
                                                                     24
  long* bad{new long};                                               25
  std::cout << *bad << std::endl;                                    26
  delete bad;                                                        27
  bad = nullptr;                                                     28
                                                                     29
  return EXIT_SUCCESS;                                               30
}                                                                    31
```

lists/programa0301.cpp

(a) Quais os resultados esperados da execução deste programa?

(b) O que a instrução da linha 5 faz? Explique todos seus detalhes.

(c) O que diferencia as instruções das linhas 15 e 25? Explique.

(d) O que acontecerá à execução do programa se descomentarmos a linha 13? Explique.

(e) O que acontecerá à execução do programa se excluirmos a instrução da linha 11 e descomentarmos a linha 13? Explique.

(f) O que acontecerá à execução do programa se descomentarmos a linha 19? Explique.

(g) O que acontecerá à execução do programa se descomentarmos a linha 23? Explique.

(h) Qual(is) problema(s) a exclusão das linhas 10, 17 ou 27 acarretaria?

2. Considere as seguintes instruções:

```
int* pointer{new (std::nothrow) int{42}};
if (pointer != nullptr)
   // do something with pointer...
```

(a) O que há de novo na alocação de memória? Explique. Sugestão: consulte `std::bad_alloc`.

(b) Qual a relação entre alocação e o teste do condicional? Como eles se complementam?

3. Considere as instruções seguintes:

```
int* ptr1{};
int* ptr2{nullptr};
```

O ponteiros `ptr1` e `ptr2` terão o mesmo valor inicial? Justifique.

4. Descreva o espaço alocado para cada ponteiro abaixo, indicando os tipos e os valores apontados:

```cpp
#include <iostream>

int main(int argc, char *argv[])
{
  int* p1       = new int;
  int* p2       = new int(5);
  int* p3       = new int{5};
  int* p4       = new int[5];
  int(* p5)[3] = new int[2][3];

  return EXIT_SUCCESS;
}
```

lists/programa0304.cpp

5. Considere o esboço de código abaixo, escrito como ponto de partida para a implementação de operações sobre um vetor de elementos de tipo `double`.

```cpp
#include <iostream>
#include <new>        // necessaria ao nothrow

struct Vector{
        int tam;            // numero de elementos
        double* elementos;  // ponteiro para elementos
};

void vector_init(Vector& v, int tam){
  // inicialize os campos do vetor v com base no tamanho tam
  // teste se a alocacao foi bem sucedida
}

void vector_free(Vector& v){
  // libere o espaco alocado para o vetor
  // lembre que a desalocacao sera feita sobre um vetor
}

void vector_read(Vector& v){
  // solicite, leia os elementos e os armazene em v
}

double vector_sum(Vector& v){
  // calcule o somatorio dos elementos armazenados em v
  // retorne o somatorio calculado
}

bool vector_find(Vector& v, int key){
  // procure a chave no vetor
  // retorne true se a chave foi encontrada, false em caso contrario
}

int main(int argc, char *argv[]){
  int n{0}, chave{0};
  Vector vetor;
```

```cpp

  // solicite a capacidade n do vetor
  // inicialize o vetor com capacidade para armazenar n elementos
  // leia os n elementos e os armazene no vetor
  // calcule e imprima o somatorio dos elementos do vetor
  // solicite chave a ser buscada
  // informe se a chave foi localizada no vetor
  // libere o espaco alocado para o vetor

  return EXIT_SUCCESS;
}
```

lists/programa0305_esboco.cpp

Complete este código de forma a obter o comportamento indicado a seguir:

```
$ ./programa0305
Quantos elementos sua lista armazenara? 5
Elemento[1]: 1
Elemento[2]: 2
Elemento[3]: 3
Elemento[4]: 4
Elemento[5]: 5
Soma dos elementos informados: 15
Informe chave de busca: 6
Chave nao localizada!
$ ./programa0305
Quantos elementos sua lista armazenara? 4
Elemento[1]: 23
Elemento[2]: 12
Elemento[3]: 67
Elemento[4]: 56
Soma dos elementos informados: 158
Informe chave de busca: 12
Chave localizada!
```

6. Considere o esboço de código abaixo, escrito como ponto de partida para a implementação de operações sobre matrizes de elementos de tipo `T`.

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

template <typename T>  // elementos de tipo T
struct Matrix{
  int linhas;     // numero de linhas da matriz
  int colunas;    // numero de colunas da matriz
  T** elementos; // elementos da matriz
};
```

```
                                                                    12
template <typename T>                                               13
void matrix_creator(Matrix<T>& m, int linhas, int colunas)          14
{                                                                   15
  // inicializar dimensoes (linhas e colunas)                       16
  // alocar as linhas da matriz                                     17
  // alocar as colunas da matriz                                    18
}                                                                   19
                                                                    20
template <typename T>                                               21
void matrix_destructor(Matrix<T>& m)                                22
{                                                                   23
  // desalocar a coluna de cada linha da matriz                     24
  // desalocar as linhas da matriz                                  25
}                                                                   26
                                                                    27
template <typename T>                                               28
void matrix_read(Matrix<T>& m)                                      29
{                                                                   30
  // ler o elemento de cada linha i e coluna j da matriz            31
}                                                                   32
                                                                    33
template <typename T>                                               34
void matrix_write(Matrix<T>& m)                                     35
{                                                                   36
  // escrever o elemento de cada linha i e coluna j da matriz       37
}                                                                   38
                                                                    39
template <typename T>                                               40
void matrix_transpose(Matrix<T>& m1, Matrix<T>& m2)                 41
{                                                                   42
  // alocar a matriz para armazenar a transposta                    43
  // realizar a transposicao de m1 em m2                            44
}                                                                   45
                                                                    46
int main(int argc, char *argv[])                                    47
{                                                                   48
  Matrix<int> m1, m2;                                               49
  int linhas, colunas;                                              50
                                                                    51
  // solicitar numero de linhas da matriz m1                        52
  // solicitar numero de colunas da matriz m1                       53
  // alocar a matriz m1                                             54
  // solicitar e ler valores da matriz m1                           55
  // escrever a matriz m1 informada                                 56
  // transpor e escrever a matriz transposta m2                     57
  // desalocar o espaco utilizado por m1 e m2                       58
                                                                    59
  return EXIT_SUCCESS;                                              60
}                                                                   61
```

lists/programa0306_esboco.cpp

Complete este código de forma a obter o comportamento indicado a seguir:

```
$ ./programa0306
Informe o numero de linhas da matriz M: 2
Informe o numero de colunas da matriz M: 4
Informar elementos da matriz M
M[0, 0]: 1
M[0, 1]: 2
M[0, 2]: 3
M[0, 3]: 4
M[1, 0]: 5
M[1, 1]: 6
M[1, 2]: 7
M[1, 3]: 8
Matriz informada:
   1   2   3   4
   5   6   7   8
Matriz transposta:
   1   5
   2   6
   3   7
   4   8
```

7. Utilizando o código da questão anterior como base, crie um programa que:
   - Solicite o número de linhas e colunas de uma matriz $M1$;
   - Leia os elementos de $M1$;
   - Solicite o número de linhas e colunas de uma matriz $M2$;
   - Leia os elementos de $M2$;
   - Verifique se é possível obter a matriz $M3 = M1 \times M2$. Neste caso, computar e escrever $M1$, $M2$ e $M3$ em tela, nesta ordem. Senão, informar que não existe $M3 = M1 \times M2$.
8. Considere a seguinte listagem:

```
/** @file list1301.cpp */
/** Listing 13-1. Copying Integers from a File to Standard Output */
#include <cstdio>
#include <fstream>
#include <iostream>

int main()
{
  std::ifstream in("list1301.txt");
  if (not in)
    std::perror("list1301.txt");
  else
  {
    int x(0);
    while (in >> x)
```

```
      std::cout << x << '\n';
    in.close();
  }
}
```

lists/list1301.cpp

Este programa lê inteiros de um arquivo chamado `list1301.txt` e os escreve, um por linha, na saída padrão. Se o arquivo não puder ser aberto, ele imprimirá uma mensagem de erro.

(a) Execute o programa sem que o arquivo de entrada exista. Qual a mensagem de erro produzida?

(b) Crie um arquivo de entrada e teste o programa. Depois disso, proteja o arquivo contra leitura. Qual mensagem de erro o programa exibirá neste caso?

9. Modifique o programa anterior de forma que ele escreva os números lidos para um arquivo de saída. Nomeie o arquivo de entrada como `list1303.in` e o arquivo de saída como `list1303.out`. Lembre-se de verificar se os arquivos podem ser abertos com sucesso e não se esqueça de fechá-los. Após fechar o arquivo de saída, verifique se a ação foi bem sucedida. No caso do arquivo de entrada, o fechamento é obrigatório? Nomeie o programa como `list1303.cpp`.

10. Crie o programa `q10.cpp` para ler um arquivo com uma série de linhas no seguinte formato:

    `palavra : n`

    onde `palavra` é uma string e `n` um inteiro. Seu programa deve gerar um novo arquivo contendo `n` ocorrências, separadas por espaços, de cada palavra `p` do arquivo de entrada. Utilize uma função auxiliar que, dadas uma palavra e seu número de ocorrências, escreva o resultado correspondente no arquivo de saída, também passado como parâmetro a esta função. Veja, no exemplo de execução abaixo, como seu programa deve se comportar.

```
$ more input.txt
bacharelado : 5
tecnologia : 5
informacao : 5
$ ./q10 input.txt output.txt
$ more output.txt
bacharelado bacharelado bacharelado bacharelado bacharelado tecnologia
tecnologia tecnologia tecnologia tecnologia informacao informacao informacao
informacao informacao
```