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 8: criação de namespaces e bibliotecas, manipulação de exceções.

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: 28 de novembro 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 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. A listagem abaixo mostra a utilização de membros de dados estáticos na implementação de um gerador de identificadores. Para gerar cada identificador, o gerador utiliza um prefixo, um valor base e um contador serial. Como esta listagem tem por objetivo inicial apenas mostrar membros de dados estáticos, o prefixo foi simplificado e sempre é inicializado com o valor 1.

```
1 #include <iostream>
2 #include <climits>
3
4 class generate_id{
5 public:
```

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

```
6   generate_id() = delete;
7   generate_id(short value) : base_{value} {}
8   long next();
9 private:
10  short base_;
11  static short counter_;
12  static short const prefix_;
13 };
14
15 short generate_id::counter_{0};
16
17 // Switch to random-number as the initial prefix for production code.
18 // short const generate_id::prefix_(static_cast<short>(std::rand()));
19 short const generate_id::prefix_{1};
20
21 long generate_id::next(){
22   if (counter_ == SHRT_MAX)
23     counter_ = 0;
24   else
25     ++counter_;
26   return static_cast<long>(prefix_) * base_ + counter_;
27 }
28
29 int main() {
30   generate_id generator1{100}, generator2{200}; // Create two ID generators
31
32   std::cout << "Generator with base value 100: " << std::endl;
33   for (int i{0}; i != 10; ++i)
34     std::cout << generator1.next() <<  std::endl;
35
36   std::cout << "Generator with base value 200: " << std::endl;
37   for (int i{0}; i != 10; ++i)
38     std::cout << generator2.next() <<  std::endl;
39 }
```

lists/list4009.cpp

Pede-se:

(a) As linhas 11 e 12 definem os membros de dados estáticos da classe `generate_id`. Quando estes membros de dados são alocados e desalocados? Consulte a bibliografia recomendada.

(b) O que as linhas 15 e 19 demonstram sobre a qualificação de acesso e inicialização de membros estáticos?

(c) Teremos versões individuais dos membros estáticos em objetos `generate_id`? Execute o programa e comprove sua resposta.

(d) Podemos inicializar `counter_` na linha 11, em vez de na linha 15?

(e) Podemos inicializar `prefix_` na linha 12, em vez de na linha 19? Neste caso, podemos comentar a linha 19 sem modificar o comportamento do programa?

2. Em C++ podemos definir qualquer função ou entidade global em qualquer arquivo-fonte. Cada arquivo-fonte pode ser compilado em um arquivo-objeto separadamente, desde que informemos

ao compilador onde os nomes referenciados estão declarados. Estes arquivos objetos podem ser linkados posteriormente, desde que o linkeditor possa encontrar uma definição para cada nome referenciado. Abaixo, reescrevemos a listagem `list4009.cpp` definindo a classe `generate_id` no arquivo `generate_id.cpp` e o programa principal no arquivo `main.cpp`.

```cpp
#include <climits>

class generate_id{
public:
  generate_id() = delete;
  generate_id(short value) : base_{value} {}
  long next();
private:
  short base_;
  static short counter_;
  static short const prefix_;
};

short generate_id::counter_{0};

// Switch to random-number as the initial prefix for production code.
// short const generate_id::prefix_(static_cast<short>(std::rand()));
short const generate_id::prefix_{1};

long generate_id::next(){
  if (counter_ == SHRT_MAX)
    counter_ = 0;
  else
    ++counter_;
  return static_cast<long>(prefix_) * base_ + counter_;
}
```

lists/q02/generate_id.cpp

Podemos compilar `generate_id.cpp`, mas não linká-lo pois precisamos da função `main`.

```cpp
#include <iostream>

class generate_id{
public:
  generate_id() = delete;
  generate_id(short value) : base_{value} {}
  long next();
private:
  short base_;
  static short counter_;
  static short const prefix_;
};

int main() {
  generate_id generator1{100}, generator2{200}; // Create two ID generators

  std::cout << "Generator with base value 100: " << std::endl;
  for (int i{0}; i != 10; ++i)
```

```
19      std::cout << generator1.next() <<  std::endl;
20
21    std::cout << "Generator with base value 200: " << std::endl;
22    for (int i{0}; i != 10; ++i)
23      std::cout << generator2.next() <<  std::endl;
24  }
```

lists/q02/main.cpp

Como `main` utiliza a classe `generate_id`, seu arquivo precisa da definição da classe `generate_id`. Pede-se:

(a) Compile (sem linkar) o arquivo `generate_id.cpp` utilizando o seguinte comando:

```
g++ -pedantic -std=c++11 -c generate_id.cpp
```

Qual arquivo foi gerado?

(b) Compile (sem linkar) o arquivo `main.cpp` utilizando o seguinte comando:

```
g++ -pedantic -std=c++11 -c main.cpp
```

Qual arquivo foi gerado?

(c) Linke os arquivos objeto gerados nos itens anteriores utilizando o seguinte comando:

```
g++ -pedantic -std=c++11 main.o generate_id.o -o main
```

O executável gerado tem o mesmo comportamento do programa `list4009`?

(d) Substitua os passos descritos nos itens (b) e (c) pelo comando abaixo:

```
g++ -pedantic -std=c++11 main.cpp -o main generate_id.o
```

O executável gerado tem o mesmo comportamento do programa `list4009`?

(e) Crie um arquivo `makefile` para seus programas. Consulte esta referência *aqui*.

(f) Note que ambos os arquivos-fonte contêm uma definição idêntica da classe `generate_id`. O que acontecerá se a definição mudar apenas em um destes arquivos? Para testar, renomeie o método `next()` como `next_id()` em `main.cpp` (linhas 7, 19 e 23) e tente recompilar.

3. Lembrando que o compilador precisa apenas da declaração das entidades globais referenciadas para criar os códigos-objeto, podemos evitar os problemas associados ao item (f) da questão anterior colocando a declaração destas entidades em um arquivo à parte e, então, utilizar a diretiva `#include` para incluir estas declarações em cada arquivo-fonte que precisa destas declarações. Desta forma, garantimos que o compilador sempre verá a mesma declaração, independente de qual código-fonte está sendo considerado. *Arquivos de cabeçalho* são utilizados para armazenar as declarações comuns a diversos arquivos-fonte e, por convenção, utilizam as extensões `.h`, `.hh`, `.hxx` ou `.hpp`. Daqui em diante, utilizaremos a extensão `.hpp` para arquivos de cabeçalho. A seguir, veja a nova estruturação dos códigos mostrados na questão anterior:

```
1 #include <climits>
2
3 class generate_id{
4 public:
5   generate_id() = delete;
6   generate_id(short value) : base_{value} {}
7   long next();
8 private:
```

```
9   short base_;
10  static short counter_;
11  static short const prefix_;
12 };
```

lists/q03/generate_id.hpp

```
1  #include "generate_id.hpp"
2
3  short generate_id::counter_{0};
4
5  // Switch to random-number as the initial prefix for production code.
6  // short const generate_id::prefix_(static_cast<short>(std::rand()));
7  short const generate_id::prefix_{1};
8
9  long generate_id::next(){
10   if (counter_ == SHRT_MAX)
11     counter_ = 0;
12   else
13     ++counter_;
14   return static_cast<long>(prefix_) * base_ + counter_;
15 }
```

lists/q03/generate_id.cpp

```
1  #include <iostream>
2  #include "generate_id.hpp"
3
4  int main() {
5    generate_id generator1{100}, generator2{200}; // Create two ID generators
6
7    std::cout << "Generator with base value 100: " << std::endl;
8    for (int i{0}; i != 10; ++i)
9      std::cout << generator1.next() <<  std::endl;
10
11   std::cout << "Generator with base value 200: " << std::endl;
12   for (int i{0}; i != 10; ++i)
13     std::cout << generator2.next() <<  std::endl;
14 }
```

lists/q03/main.cpp

Pede-se:

(a) Note a diferença entre as inclusões de bibliotecas padrão e do usuário (linhas 1 e 2 de `main.cpp`, respectivamente). Qual o tratamento dado pelo compilador nestes casos?

(b) Podemos utilizar o mesmo arquivo makefile para gerar o executável? Verifique se o novo programa preserva o comportamento da versão original.

(c) Note que em `generate_id.hpp` temos a inclusão de um arquivo de cabeçalho, ou seja, um arquivo de cabeçalho pode ter suas próprias diretivas de inclusão.

(d) O que acontecerá se um arquivo de cabeçalho for incluído mais de uma vez, *mesmo que indiretamente por meio de outras inclusões*, por um mesmo arquivo fonte?

4. Para assegurar que o compilador veja apenas uma vez a declaração de objetos globais, ainda que um mesmo arquivo-fonte inclua seu cabeçalho mais que uma vez, devemos utilizar diretivas adicionais. Para isso, escolha um nome para identificar inequivocamente seu arquivo de cabeçalho e utilize este nome para controlar a compilação. Veja o exemplo a seguir:

```
#ifndef GENERATE_ID_HPP_
#define GENERATE_ID_HPP_

#include <climits>

class generate_id{
public:
  generate_id() = delete;
  generate_id(short value) : base_{value} {}
  long next();
private:
  short base_;
  static short counter_;
  static short const prefix_;
};

#endif
```

lists/q04/generate_id.hpp

Explique o funcionamento destas diretivas.

5. Considere as seguintes listagens, com anotações ***Doxygen***:

```
#ifndef VITAL_STATS_HPP_
#define VITAL_STATS_HPP_

#include <iosfwd>
#include <string>

class vital_stats{
public:
  /// Constructor. Initialize everything to zero or other "empty" value.
  vital_stats() : height_{0}, weight_{0}, bmi_{0}, sex_{'?'}, name_{}
  {}
  /// Get this record, overwriting the data members.
  /// Error-checking omitted for brevity.
  /// @param in the input stream
  /// @param num a serial number, for use when prompting the user
  /// @return true for success or false for eof or input failure
  bool read(std::istream& in, int num);
  /// Print this record to @p out.
  /// @param out the output stream
  /// @param threshold mark records that have a BMI >= this threshold
  void print(std::ostream& out, int threshold) const;
  /// Return the BMI.
  int bmi() const { return bmi_; }
  /// Return the height in centimeters.
  int height() const { return height_; }
```

```
26   /// Return the weight in kilograms.
27   int weight() const { return weight_; }
28   /// Return the sex: 'M' for male or 'F' for female
29   char sex() const { return sex_; }
30   /// Return the person's name.
31   std::string const& name() const { return name_; }
32 private:
33   /// Return BMI, based on height_ and weight_
34   /// This is called only from read().
35   int compute_bmi() const;
36   int height_;
37   int weight_;
38   int bmi_;
39   char sex_;
40   std::string name_;
41   ///< height in centimeters
42   ///< weight in kilograms
43   ///< Body-mass index
44   ///< 'M' for male or 'F' for female
45   ///< Person's name
46 };
47
48 #endif
```

lists/q05/vital_stats.hpp

```
1  #include <iomanip>
2  #include <iostream>
3  #include <limits>
4  #include <locale>
5  #include <string>
6
7  #include "vital_stats.hpp"
8
9  /// Skip the rest of the input line.
10 /// @param in the input stream
11 void skip_line(std::istream& in){
12   in.ignore(std::numeric_limits<int>::max(), '\n');
13 }
14
15 int vital_stats::compute_bmi()
16 const {
17    return static_cast<int>(weight_ * 10000 / (height_ * height_) + 0.5);
18 }
19
20 bool vital_stats::read(std::istream& in, int num){
21   std::cout << "Name " << num << ": ";
22   if (not std::getline(in, name_))
23     return false;
24   std::cout << "Height (cm): ";
25   if (not (in >> height_))
26     return false;
27   skip_line(in);
```

```
28   std::cout << "Weight (kg): ";
29   if (not (in >> weight_))
30     return false;
31   skip_line(in);
32   std::cout << "Sex (M or F): ";
33   if (not (in >> sex_))
34     return false;
35   skip_line(in);
36   sex_ = std::toupper(sex_, std::locale{});
37   bmi_ = compute_bmi();
38  return true;
39 }
40
41 void vital_stats::print(std::ostream& out, int threshold)
42 const {
43   out << std::setw(6) << height_
44       << std::setw(7) << weight_
45       << std::setw(3) << sex_
46       << std::setw(6) << bmi_;
47   if (bmi_ >= threshold)
48     out << '*';
49   else
50     out << ' ';
51   out << ' ' << name_ << '\n';
52 }
```

lists/q05/vital_stats.cpp

```
1 #include <iostream>
2 #include <vector>
3
4 #include "vital_stats.hpp"
5
6 /// Reads and prints the vital stats of two persons.
7 int main(){
8   std::vector<vital_stats> stats{};
9   vital_stats s{};
10
11   for (int i{0}; i < 2; i++){
12     s.read(std::cin, i);
13     stats.push_back(s);
14   }
15
16   for (auto e: stats)
17     e.print(std::cout, 25);
18
19   return 0;
20 }
```

lists/q05/main.cpp

Pede-se:

(a) Consulte a documentação do Doxygen *aqui*. Certifique-se de ter entendido todo o código.

(b) Compile estas listagens via makefile. Os resultados da execução foram os esperados?

(c) Gere a documentação correspondente a estas listagens utilizando o Doxygen.

6. Considere as seguintes listagens:

```hpp
#ifndef MATH_HPP_
#define MATH_HPP_

namespace math{

double area(double);

}

#endif
```

lists/q06/math.hpp

```cpp
#include "math.hpp"

extern double pi;

namespace math{

double area(double r){ return pi * r * r; }

}
```

lists/q06/math.cpp

```cpp
#include <iostream>

#include "math.hpp"

// More digits that typical implementations of double support.
double pi{3.14159265358979323846264338327};

int main(){
  double r{};

  std::cout << "Inform the radius of the circle: ";
  std::cin >> r;
  std::cout << "Area of the circle: " << math::area(r) << std::endl;

  return 0;
}
```

lists/q06/main.cpp

Pede-se:

(a) O que faz a linha 3 de `math.cpp`? Explique o funcionamento de variáveis `extern`.

(b) Observe as listagens `math.hpp` e `math.cpp`. Explique como criar namespaces.

(c) Quais as vantagens de criar namespaces para suas bibliotecas?

(d) Compile estas listagens via makefile. Os resultados da execução foram os esperados?

7. Considere a seguinte listagem, onde introduzimos o ***tratamento de exceções***:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6    string line{};
7    while (getline(cin, line)){
8      try{
9        line.at(10) = ' '; // can throw out_of_range
10       if (line.size() < 20)
11         line.append(line.max_size(), '*'); // can throw length_error
12       for (string::size_type size(line.size());
13            size < line.max_size(); size = size * 2){
14         line.resize(size); // can throw bad_alloc
15       }
16       line.resize(line.max_size()); // can throw bad_alloc
17       cout << "okay\n";
18     }
19     catch (out_of_range const& ex){
20       cout << ex.what() << '\n';
21       cout << "string index (10) out of range.\n";
22     }
23     catch (length_error const& ex){
24       cout << ex.what() << '\n';
25       cout << "maximum string length (" << line.max_size() << ") exceeded.\n";
26     }
27     catch (exception const& ex){
28       cout << "other exception: " << ex.what() << '\n';
29     }
30     catch (...){
31       cout << "Unknown exception type. Program terminating.\n";
32       abort();
33     }
34   }
35   return 0;
36 }
```

lists/q07/list4503.cpp

(a) Consulte as referências bibliográficas da disciplina a respeito do tratamento de exceções. Veja também os exemplos disponibilizados nos sites *cplusplus.com* e *tutorialspoint*. Após a leitura destas referências, assegure-se de compreender toda a listagem `list4503.cpp`.

(b) Quando uma exceção lançada combina com a de um manipulador? A ordem dos manipuladores é importante? Explique o funcionamento geral de um bloco `try-catch`.

(c) Explique o funcionamento do manipulador da linha 30.

(d) Quais as situações que podem levar o código da listagem `list4503.cpp` a lançar exceções?

(e) Utilize um makefile para compilar esta listagem. Execute o programa com strings de tamanhos variados e certifique-se de compreender os resultados produzidos pelo programa.

8. Quando um programa lança uma exceção, o fluxo de controle normal é interrompido e o mecanismo de manipulação de exceções assume o controle. O objeto que representa a exceção lançada é copiado para uma região de armazenamento segura e o mecanismo de manipulação de exceções passa a buscar por um bloco `try-catch` na pilha de execução. Ao encontrar um bloco `try-catch`, o mecanismo verifica os tipos dos manipuladores em busca de compatibilidade. Se nenhuma compatibilidade for identificada, o mecanismo busca pelo próximo bloco `try-catch` abaixo na pilha de execução. Este processo continua até que um manipulador compatível seja encontrado ou a pilha se esgote. Se um manipulador compatível é encontrado, todos os frames previamente investigados são desempilhados da pilha de execução, sendo invocados os destrutores de todos os objetos locais dos frames desempilhados, até que se chegue ao frame onde o manipulador compatível se encontra. Este processo de desempilhamento é conhecido como *unwinding*. Após o *unwinding*, o objeto da exceção lançada inicializa o objeto da exceção do manipulador e o corpo do `catch` é executado. Após a execução do `catch`, o objeto da exceção é liberado e a execução continua na instrução seguinte ao fim do bloco `try-catch` correspondente. Caso nenhum manipulador compatível seja identificado, o mecanismo de manipulação de exceções chama a função `std::terminate` para abortar a execução do programa. Considere a seguinte listagem a seguir, que utilizaremos para ilustrar esse comportamento:

```
1  #include <exception>
2  #include <iostream>
3  #include <string>
4
5  /// Make visual the construction and destruction of objects.
6  class visual
7  {
8  public:
9     visual(std::string const& what)
10    : id_{serial_}, what_{what}
11    {
12      ++serial_;
13      print("");
14    }
15    visual(visual const& ex)
16    : id_{ex.id_}, what_{ex.what_}
17    {
18      print("copy ");
19    }
20    ~visual()
21    {
22      print("~");
23    }
24    void print(std::string const& label)
25    const
26    {
27      std::cout << label << "visual(" << what_ << ": " << id_ << ")\n";
28    }
29 private:
30   static int serial_;
31   int const id_;
```

```
32   std::string const what_;
33 };
34
35 int visual::serial_{0};
36
37 void count_down(int n)
38 {
39   std::cout << "start count_down(" << n << ")\n";
40   visual v{"count_down local"};
41   try
42   {
43     if (n == 3)
44       throw visual("exception");
45     else if (n > 0)
46       count_down(n - 1);
47   }
48   catch (visual ex)
49   {
50     ex.print("catch ");
51     throw;
52   }
53   std::cout << "end count_down(" << n << ")\n";
54 }
55
56 int main()
57 {
58   try
59   {
60     count_down(2);
61     count_down(4);
62   }
63   catch (visual const ex)
64   {
65     ex.print("catch ");
66   }
67   std::cout << "All done!\n";
68 }
```

lists/q08/list4504.cpp

Compile esta listagem e certifique-se de compreender os resultados exibidos.

9. Considere as seguintes listagens, onde exceções personalizadas são criadas:

```cpp
#include <iostream>
#include <exception>

using namespace std;

class MyException : public exception {
public:
   const char * what () const noexcept {
      return "C++ Exception";
   }
};

int main() {
   try
   {
      throw MyException();
   }
   catch(MyException& e)
   {
      std::cout << "MyException caught" << std::endl;
      std::cout << e.what() << std::endl;
   }
   catch(std::exception& e)
   {
      //Other errors
   }
   return 0;
}
```

lists/q09/my_exception.cpp

```cpp
#include <stdexcept>
#include <string>

class rational {
public:
  class zero_denominator : public std::logic_error {
  public:
    zero_denominator(std::string const& what_arg) : logic_error{what_arg} {}
  };
  rational() : rational{0} {}
  rational(int num) : numerator_{num}, denominator_{1} {}
  rational(int num, int den) : numerator_{num}, denominator_{den} {
    if (denominator_ == 0)
      throw zero_denominator{"zero denominator"};
    reduce();
  }
  // Omitted for brevity
};
```

lists/q09/zero_denominator.cpp

Com base nestes exemplos, crie uma nova exceção que seja útil ao projeto de seu grupo.