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 6: herança, métodos virtuais/classes abstratas, polimorfismo.

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: 19 de outubro 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 a listagem a seguir, onde classes simples representam itens do acervo de uma biblioteca. Embora diversos tipos de itens possam ser encontrados em uma biblioteca (livros, revistas, filmes, entre outros), esta listagem considera apenas dois tipos de trabalho: livros e periódicos. Note que a definição de uma classe derivada é bastante parecida com a definição de classe que já conhecemos, exceto pela inclusão do nível de acesso e nome da classe base. Por enquanto, ignore as palavras-chave `virtual` e `override`, voltaremos a elas em breve.

```
/** A class hierarchy for representing items of a library. **/
#include <iostream>
#include <string>
```

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

```
                                                                                4
using namespace std;                                                            5
                                                                                6
class work{                                                                     7
public:                                                                         8
  work() = default;                                                             9
  work(work const&) = default;                                                  10
  work(string const& id, string const& title) : id_{id}, title_{title} {}       11
  virtual ~work() {}                                                            12
  string const& id()    const { return id_; }                                   13
  string const& title() const { return title_; }                                14
  virtual void print(ostream&) const {}                                         15
private:                                                                        16
  string id_;                                                                   17
  string title_;                                                                18
};                                                                              19
                                                                                20
class book : public work{                                                       21
public:                                                                         22
  book() : work{}, author_{}, pubyear_{0} {}                                    23
  book(string const& id, string const& title, string const& author, int pubyear) 24
  : work{id, title}, author_{author}, pubyear_{pubyear} {}                      25
  string const& author() const { return author_; }                              26
  int pubyear()          const { return pubyear_; }                             27
  void print(ostream& out) const override {                                     28
    out << author() << ", " << title() << ", " << pubyear() << ".";             29
  }                                                                             30
private:                                                                        31
  string author_;                                                               32
  int pubyear_; ///< year of publication                                       33
};                                                                              34
                                                                                35
class periodical : public work{                                                 36
public:                                                                         37
  periodical() : work{}, volume_{0}, number_{0}, date_{} {}                     38
  periodical(string const& id, string const& title, int volume,                 39
             int number, string const& date)                                    40
  : work{id, title}, volume_{volume}, number_{number}, date_{date} {}           41
  int volume() const { return volume_; }                                        42
  int number() const { return number_; }                                        43
  string const& date() const { return date_; }                                  44
  void print(ostream& out) const override {                                     45
    out << title() << ", "                                                      46
        << volume() << '(' << number() << "), " << date() << ".";               47
  }                                                                             48
private:                                                                        49
  int volume_;        ///< volume number                                        50
  int number_;        ///< issue number                                         51
  string date_; ///< publication date                                           52
};                                                                              53
                                                                                54
```

```
int main(){}
```

lists/list3601.cpp

(a) O que significam os níveis de acesso das linhas 21 e 36?

(b) Nas linhas 23 e 25, assim como nas linhas 38 e 41, fazemos chamadas a construtores da classe base. O que acontecerá se não invocarmos estes construtores explicitamente?

(c) Na linha 9, como o compilador se comportará se trocarmos `default` por `delete`? E se também apagarmos as chamadas ao construtor padrão das linhas 23 e 38?

(d) Nas linhas 29 e 46 o método `title()` é invocado. Onde está a definição deste método?

2. Considere a listagem seguinte:

```
#include <iostream>
class base{
public:
  base() { std::cout << "base\n"; }
  ~base() { std::cout << "~base\n"; }
};

class middle : public base{
public:
  middle() { std::cout << "middle\n"; }
  ~middle() { std::cout << "~middle\n"; }
};

class derived : public middle{
public:
  derived() { std::cout << "derived\n"; }
  ~derived() { std::cout << "~derived\n"; }
};

int main() {
  derived d;
}
```

lists/list3602.cpp

Qual o resultado da execução deste programa? O que ele nos mostra sobre a ordem de inicialização das classes base e derivadas? E sobre a ordem de execução dos destrutores?

3. Considere a listagem seguinte:

```
#include <iostream>
#include <vector>

class base{
public:
  base(int value) : value_{value} { std::cout << "base(" << value << ")\n"; }
  base() : base{0} { std::cout << "base()\n"; }
  base(base const& copy)
  : value_{copy.value_}
  { std::cout << "copy base(" << value_ << ")\n"; }
  ~base() { std::cout << "~base(" << value_ << ")\n"; }
```

```
  int value() const { return value_; }
  base& operator++()
  {
    ++value_;
    return *this;
  }
private:
  int value_;
};

class derived : public base{
public:
  derived(int value): base{value} { std::cout << "derived(" << value << ")\n"; }
  derived() : base{} { std::cout << "derived()\n"; }
  derived(derived const& copy)
  : base{copy}
  { std::cout << "copy derived(" << value() << "\n"; }
  ~derived() { std::cout << "~derived(" << value() << ")\n"; }
};

derived make_derived(){
  return derived{42};
}

base increment(base b){
  ++b;
  return b;
}

void increment_reference(base& b){
  ++b;
}

int main() {
  derived d{make_derived()};
  base b{increment(d)};
  increment_reference(d);
  increment_reference(b);
  derived a(d.value() + b.value());
}
```

lists/list3704.cpp

Quais os resultados esperados e os obtidos da execução deste programa?

4. Considere a listagem seguinte:

```
class base{
public:
  base(int v) : value_{v} {}
  int value() const { return value_; }
private:
  int value_;
};
```

```
                                                                    8
class derived : base{                                               9
public:                                                             10
  derived() : base{42} {}                                           11
};                                                                  12
                                                                    13
int main() {                                                        14
  base b{42};                                                       15
  int x{b.value()};                                                 16
  derived d{};                                                      17
  int y{d.value()};                                                 18
}                                                                   19
```

lists/list3705.cpp

Explique a razão dos erros apontados pelo compilador ao processar este programa.

5. Na listagem `list3601.cpp`, apresentada na primeira questão, os métodos `void print(ostream&) const override` imprimem informações sobre os trabalhos de acordo com o seguinte formato:

   - Livros*:* ***autor***, ***título***, ***ano***
   - Periódicos: ***título***, ***volume(número)***, ***data***

   Considere, agora, a listagem `list3802.cpp`, que estende esta primeira listagem com a função `void showoff(...)` e nova função `main` (as linhas 1 a 54 foram omitidas por brevidade):

```
void showoff(work const& w){                                        55
  w.print(std::cout);                                               56
  std::cout << '\n';                                                57
}                                                                   58
                                                                    59
int main() {                                                        60
  book sc{"1", "The Sun Also Crashes", "Ernest Lemmingway", 2000};  61
  book ecpp{"2", "Exploring C++", "Ray Lischner", 2013};            62
  periodical pop{"3", "Popular C++", 13, 42, "January 1, 2000"};    63
  periodical today{"4", "C++ Today", 1, 1, "January 13, 1984"};     64
  showoff(sc);                                                      65
  showoff(ecpp);                                                    66
  showoff(pop);                                                     67
  showoff(today);                                                   68
}                                                                   69
```

lists/list3802.cpp

   (a) Note que o parâmetro da função `void showoff(...)` é do tipo `work`. Qual o resultado esperado da execução deste programa? Quais os resultados obtidos? Explique.

   (b) O que os modificadores `virtual` (linha 15) e `override` (linhas 28 e 45) da listagem `list3601.cpp` especificam? Qual a relação entre esses recursos e a definição de *funções polimórficas*? Consulte suas referências para embasar sua resposta.

   (c) Crie um operador de impressão `operator<<` para imprimir objetos `work` utilizando os métodos `... print(...)`. Salve seu código como `list3803.cpp`.

   (d) O que acontecerá se mudarmos a função `void showoff(...)` para que ela receba um parâmetro passado por valor (tornando-se `void showoff(work w)`)? Explique.

6. Com base na listagem `list3803.cpp` criada no item anterior, adicione uma classe `movie` às classes da biblioteca e salve a listagem modificada como o nome `list3804.cpp`. A classe `movie` deve representar filmes e, assim como a classe `book` e `periodical`, `movie` deve derivar de `work`. Por simplicidade, defina a classe `movie` como tendo um inteiro para representar a duração da gravação em minutos, em adição aos membros herdados de `work`. Em seguida, crie e imprima um objeto de tipo `movie`, utilizando o operador de impressão definido no item anterior.

7. Salve a listagem `list3804.cpp` criada no item anterior com o nome `list3805.cpp`. Em seguida, modifique a classe `work` para fazer seu método de impressão uma *função virtual pura* e, então, apague o método de impressão de `book`. Consulte as referências indicadas quando necessário.

   (a) Qual(is) classe(s) se torna(m) *abstrata(s)* após essas alterações?

   (b) O que acontece ao compilarmos a listagem `list3805.cpp`? Explique.

   (c) Quais restrições existem sobre *classes abstratas*?

8. Considere a listagem `list6601.cpp`:

```cpp
#include <iostream>
#include <string>

using namespace std;

class visible {
public:
    visible(string&& msg) : msg_{move(msg)} { cout << msg_ << '\n'; }
    string const& msg() const { return msg_; }
private:
    string msg_;
};

class base1 : public visible {
public:
   base1(int x) : visible{"base1 constructed"}, value_{x} {}
   int value() const { return value_; }
private:
   int value_;
};

class base2 : public visible {
public:
   base2(string const& str) : visible{"base2{" + str + "} constructed"} {}
};

class base3 : public visible {
public:
   base3() : visible{"base3 constructed"} {}
   int value() const { return 42; }
   // string const& msg() const { return msg_; }
};

class derived : public base1, public base2, public base3 {
public:
```

```
   derived(int i, string const& str) : base3{}, base2{str}, base1{i} {}          36
   int value() const { return base1::value() + base3::value(); }                 37
   string msg() const                                                            38
   {                                                                             39
     return base1::msg() + "\n" + base2::msg() + "\n" + base3::msg();            40
   }                                                                             41
};                                                                               42
                                                                                 43
int main() {                                                                     44
   derived d{42, "example"};                                                     45
}                                                                                46
```

lists/list6601.cpp

Este é nosso primeiro primeiro programa envolvendo herança múltipla. Note, na linha 34, que declaramos as classes base listando-as em uma lista separada por vírgulas, cada classe base com seu próprio especificador de acesso. A figura abaixo ilustra a hieraquia de classes:

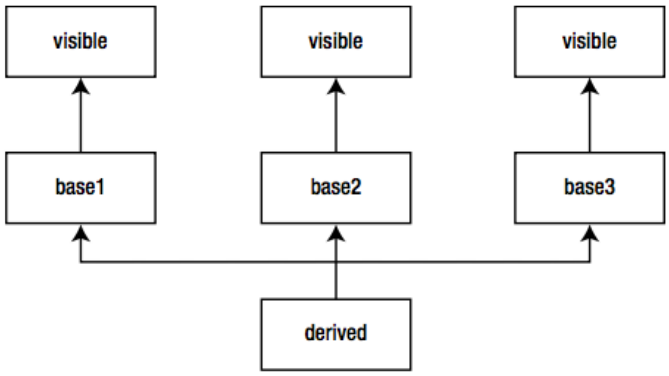

Figura 1: Diagrama UML das classes da listagem `list6601.cpp`.

Pede-se:

(a) Explique o comportamento do construtor da classe `visible`. Atenção aos detalhes.

(b) Quais erros serão introduzidos se descomentarmos a linha 31? Explique.

(c) Na linha 10, experimente mudar o nível de acesso do atributo `msg_` de `private` para `protected`. O que acontece ao descomentarmos a linha 31. Explique.

(d) Ao executar este programa, o que você deduz a respeito da ordem de execução da inicialização das classes bases utilizadas pela derivada (linha 36)?

(e) Observe como diferenciar chamadas a métodos de mesmo nome, herdados de bases distintas (linha 40). Note que os métodos `msg()` herdados co-existem, cada um em seu próprio espaço de nomes. Inclua, no programa principal, as instruções necessárias para a escrita do valor e da mensagem do objeto derivado na saída padrão. Salve seu código como `list6601B.cpp`.

9. Relacione, na tabela abaixo, os níveis de acesso a componentes e modos de herança.

Visibilidade do componente na classe base	Modo de herança	Visibilidade do componente na classe derivada
public	public	
protected		
private		
public	protected	
protected		
private		
public	private	
protected		
private		

10. Considere a listagem a seguir:

```
#include <iostream>
#include <string>

using namespace std;

class Data{
public:
    Data(int dia, int mes, int ano)  : dia_{dia}, mes_{mes}, ano_{ano} {}
    void MostreData() { cout << dia_ << "/" << mes_ << "/" << ano_ <<endl; }
private:
    int dia_;
    int mes_;
    int ano_;
};

class Pessoa{
public:
    Pessoa(string nome, Data dataNascimento)
    : nome_{nome}, dataNascimento_{dataNascimento} {}
    void MostreInfo(){
        cout << nome_ << " nasceu em ";
        dataNascimento_.MostreData();
    }
private:
    string nome_;
    Data dataNascimento_;
};


int main() {
    Pessoa pessoa{"Joao Silva", Data{1, 1, 1998}};
    pessoa.MostreInfo();
}
```

lists/funcionarios.cpp

Esta listagem consiste no início da implementação de um cadastro de funcionários. Utilizando

esta listagem como base, crie um cadastro de funcionários assumindo que:

- Além da informação de *nome* e *data de nascimento*, precisamos armazenar também o *CPF*, o *RG* e o *nome da mãe* de cada `Pessoa`. Crie classes `CPF` e `RG`;
- Todo `Funcionario` é uma `Pessoa`, mas além das informações comuns a uma `Pessoa`, precisamos armazenar as informações de *data de contratação* e *salário* de um `Funcionario`. Crie a classe `Funcionario` a partir da classe `Pessoa`. Utilize níveis de acesso e herança com cautela;
- Assuma que o `Cadastro` é uma classe que contém as informações dos funcionários de uma empresa. O número n de funcionários da empresa deve ser solicitado ao usuário, via teclado. Em seguida, você deve ler e armazenar as informações dos n funcionários da empresa em um objeto `Cadastro`. A classe `Cadastro` deve ter métodos para: *imprimir as informações de um funcionário* com base em seu CPF; *imprimir as informações completas de todos os funcionários* da empresa e *computar e imprimir o a soma dos salários* de todos os funcionários da empresa. Na classe `Cadastro`, represente os funcionários por meio de um arranjo de `Funcionarios`;
- Utilize construtores e destrutores adequados e promova a independência entre as classes.