UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
(DAELN)

Curso: Engenharia Eletrônica, **Disciplina:** Fundamentos de Programação II (IF62C), **Turma:** S__
Professores: Hermes Del Monego ( ), Jean Simão ( ), Robinson Vida ( ) **Data:** _________.

Aluno: **Código:**_____, **Início** __:__ **Fim** __:__

1ª Prova

1) $_{[1{,}5\,pt]}$ (*Poscomp*) Um ponteiro é um elemento que proporciona maior controle sobre a memória do computador, principalmente por ser utilizado em conjunto com mecanismos de alocação dinâmica de memória. Dessa forma, o domínio sobre este tipo de dado é muito importante. O código, a seguir, foi escrito na linguagem C++ e trabalha com ponteiros e estruturas dinâmicas.

```
#include <iostream>
using namespace std;
struct No{
        int Dado;
        No* Prox;
};
int main() {
        No *L, *i;
        int n;
        cin >> n;
        if (n == 0) L = NULL;
        else{
                        L = new No;
                        L->Dado = n--;
                        L->Prox = NULL;
                        for ( ; n > 0 ; )
                                {
                                        i = new No;
                                        i->Dado = n--;
                                        i->Prox = L;
                                        L = i;
                                }
                }
        while (L != NULL) {
                        cout << L->Dado << " ";
                        L = L->Prox;
                }
        return 0;
}
```

Se, durante a execução desse código, a variável n receber o valor 6, a saída do programa será:

a) 0 1 2 3 4 5 6

b) 1 2 3 4 5 6

c) 6 5 4 3 2 1

d) 6 5 4 3 2 1 0

e) 1 2 3 4 5

2) $_{[1{,}5\,pt]}$ (*Poscomp*) O encapsulamento dos dados tem como objetivo ocultar os detalhes da implementação de um determinado módulo. Em linguagens orientadas a objeto, o ocultamento de informação é tornado explícito requerendo-se que todos os métodos e atributos em uma classe tenham um nível particular de visibilidade com relação às suas subclasses e às classes clientes. Em relação aos atributos de visibilidade, assinale a alternativa correta.

a) Um atributo ou método público é visível a qualquer classe cliente e subclasse da classe a que ele pertence.

b) Um atributo ou método protegido é visível somente à classe a que ele pertence, mas não às suas subclasses ou aos seus clientes.

c) Um atributo ou método privado é visível somente às subclasses da classe a que ele pertence.

d) Um método protegido não pode acessar os atributos privados declarados na classe a que ele pertence, sendo necessária a chamada de outro método privado da classe.

e) Um método público pode acessar somente atributos públicos declarados na classe a que ele pertence.

3) $_{[2{,}0\,pt]}$ (*Poscomp*) Em linguagens orientadas a objetos, o polimorfismo refere-se à ligação tardia de uma chamada a uma ou várias implementações diferentes de um método em uma hierarquia de herança. Neste contexto, considere as seguintes classes descritas na Linguagem C++ na **Figura A**.

// Figura A

```
#include <iostream>
using namespace std;
class PosComp1
{
public:
        int Calcula()
        {
                return 1;
        };
};
class PosComp2 : public PosComp1
{
public:
        virtual int Calcula()
        {
                return 2;
        }
};

class PosComp3 : public PosComp2
{
public:
        int Calcula()
        {
                return 3;
        }
};
```

// Figura B

```
int main()
{
        int Result=0;
        PosComp1 *Objs[3];
        Objs[0] = new PosComp1();
        Objs[1] = new PosComp2();
        Objs[2] = new PosComp3();
        for (int i=0; i<3; i++)
                Result += Objs[i]->Calcula();
        cout << Result << endl;
        return 0;
}
```

Se estas classes forem utilizadas a partir do programa a da **Figura B**.

a) 0

b) 3

c) 5

d) 6

e) 9

4) $_{[1{,}5\,pt]}$ (*Poscomp*) O mecanismo de herança, no paradigma da programação orientada a objetos, é uma forma de reutilização de software na qual uma nova classe é criada, absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas. Considere as seguintes classes descritas na linguagem C++ e mostradas na **Figura A**.

// Figura A

```
#include <iostream>
using namespace std;
class A
{
protected:
        int v;
public:
        A()
        {
                v = 0;
        };
        void m1()
        {
                v += 10;
                m2();
        };
        void m2()
        {
                v += 20;
        };
        int getv()
        {
                return v;
        };
};
class B : public A
{
public:
        void m2()
        {
                v += 30;
        };
};
```

// Figura B

```
int main()
{
        B *Obj = new B();
        Obj->m1();
        Obj->m2();
        cout << Obj->getv() << endl;
        return 0;
}
```

Se essas classes forem utilizadas a partir do programa da **Figura B**. Qual seria a saida?

a) 30

b) 40

c) 50

d) 60

e) 70

5) $_{[1{,}5\,pt]}$ (*Poscomp*) A UML (*Unified Modeling Language*) é uma linguagem padrão para a elaboração da estrutura de projetos que pode ser empregada para a visualização, a especificação, a construção e a documentação de artefatos. No contexto da UML, um relacionamento é uma conexão entre itens, representado graficamente como um caminho, com tipos diferentes de linhas para diferenciar os tipos de relacionamento. Com base no enunciado e nos conhecimentos sobre o tema, correlacione os tipos de relacionamentos e suas respectivas descrições.

(I)	É um relacionamento de utilização, determinando que um item usa as informações e serviços de outro item, mas não necessariamente o inverso	(A)	Associação
(II)	É um relacionamento entre itens gerais e tipos mais específicos desses itens.	(B)	Dependência
(III)	É um relacionamento estrutural que especifica objetos de um item conectados a objetos de outro item. A partir deste relacionamento, é possível navegar do objeto de uma classe até o objeto de outra classe e vice-versa.	(C)	Generalização

Escolha a alternativa onde apresenta o resultado correto.

a) I-A; II-B; III-C.

b) I-B; II-A; III-C.

c) I-B; II-C; III-A.

d) I-C; II-B; III-A.

e) I-C; II-A; III-B.

6) $_{[2{,}0\,pt]}$ Elabore um diagrama de classe em UML com:

- Classe Abstrata *Carta* com o método virtual puro *imprime_carta*.
- Classes *Naipe* e *Curinga* derivadas de *Carta*.
- Classe *Baralho*.
- A instância de *Baralho* relaciona-se com um número desconhecido de objetos de classes derivadas da classe de *Carta*. Para tal pode-se incluir na instância de *Baralho* tanto os endereços de objetos da classe *Naipe* quanto os endereços de objetos da classe *Curinga* sem que a instância da classe de *Baralho* os diferencie. Por fim, a instância de *Baralho* poderá listar suas cartas por meio do método *imprime_carta*.