Suporte a programação orientada a objeto em C++

- Característica gerais
  - Baseado em C e em SIMULA 67
  - Primeira linguagem de programação OO amplamente utilizada
  - Dois sistemas de tipo: imperativo e OO
  - Objetos podem ser estáticos, dinâmicos na pilha ou dinâmicos no heap
  - Desalocação explícita usando o operador `delete`
  - Destrutores
  - Mecanismo de controle de acesso elaborado

Suporte a programação orientada a objeto em C++

- Herança
  - Tipos de controle de acesso aos membros
    - `private`, `protected` e `public`
  - Um classe não precisa ter uma classe pai

Suporte a programação orientada a objeto em C++

- Herança
  - Todos os objetos precisam ser inicializado antes de serem usados, no caso de subclasse, os membros herdados precisam ser inicializados quando uma instância da subclasse é criada
  - Uma subclasse não precisa ser um subtipo
    - Derivação pública: os membros públicos e protegidos são membros públicos e protegidos na subclasse
    - Derivação privada: todos os membros públicos e protegidos são membros privados na subclasse

Suporte a programação orientada a objeto em C++

- Herança
  - Suporta herança múltipla
    - Se dois membros herdados tem o mesmo nome, eles podem ser acessados usando o operador de resolução de escopo (`::`)
  - Um método da subclasse precisa ter os mesmos parâmetros do método da classe pai para sobrescrevê-lo. O tipo de retorno tem que ser o mesmo ou um tipo derivado (público)

```
class single_linked_list {
  private:
    class node {
      public:
        node *link;
        int contents;
    };
    node *head;
  public:
    single_linked_list() {head = 0;};
    void insert_at_head(int);
    void insert_at_tail(int);
    int remove_at_head();
    bool empty();
};
```

```
// Como stack é uma derivação pública, todos os
// métodos públicos da classe single_linked_list
// também são públicos em stack, o que deixa a
// classe com métodos públicos indesejados
// (insert_at_head, insert_at_tail e remove_at_head)

class stack: public single_linked_list {
  public:
    stack() {}
    void push(int value) {
      insert_at_head(value);
    }
    int pop() {
      return remove_at_head();
    }
};
```

```
// stack2 é uma derivação privada de
// single_linked_list, portanto os membros
// públicos e protegidos herdados de
// single_linked_list são privados em stack2.
// stack2 tem que redefinir a visibilidade do
// membro empty para torná-lo público.

class stack2: private single_linked_list {
  public:
    stack2() {}
    void push(int value) {
      insert_at_head(value);
    }
    int pop() {
      return remove_at_head();
    }
    single_linked_list::empty;
};
```

```
// Uma alternativa mais interessante é usar composição,
// o que permite que uma pilha possa ser
// definida com qualquer implementação de lista.
class stack3 {
  private:
    list *li;
  public:
    stack3(list *l) : li(l) {}
    void push(int value) {
      li->insert_at_head(value);
    }
    int pop() {
      return li->remove_at_head();
    }
    boolean empty() {
      return li->empty();
    }
};
```

Suporte a programação orientada a objeto em C++

- Vinculação dinâmica
  - Um método pode ser definido como virtual, que significa que ele será vinculado dinamicamente quando chamado em uma variável polimórfica
  - Funções virtuais puras não têm definição
  - Uma classe que tem pelo menos um método virtual puro é uma classe abstrata

```
class Shape {
  public:
    virtual void name() = 0;
};
class Rectangle: public Shape {
  public:
    void name() {
        printf("Rectangle\n");
    }
    void code() {
        printf("R\n");
    }
};
class Square: public Rectangle {
  public:
    void name() {
        printf("Square\n");
    }
    void code() {
        printf("S\n");
    }
};
```

O que será impresso?

```
Shape* s = new Rectangle();
s->name();
...
s = new Square();
s->name();
```

```
Rectangle
Square
```

name é um método virtual e portanto é vinculado dinamicamente ao método da classe **instanciada** referenciada por r.

```
class Shape {
  public:
    virtual void name() = 0;
};
class Rectangle: public Shape {
  public:
    void name() {
        printf("Rectangle\n");
    }
    void code() {
        printf("R\n");
    }
};
class Square: public Rectangle {
  public:
    void name() {
        printf("Square\n");
    }
    void code() {
        printf("S\n");
    }
};
```

O que será impresso?

```
Rectangle *r = new Rectangle();
r->code();
...
r = new Square();
r->code();
```

```
R
R
```

code **não** é um método virtual e portanto é vinculado estaticamente ao método do tipo declarado de `r`.

Suporte a programação orientada a objeto em C++

- Avaliação
  - C++ fornece muitas formas de controle de acesso (diferente de Smalltalk)
  - C++ fornece herança múltipla
  - Em C++, é necessário definir em tempo de projeto quais métodos serão vinculados estaticamente e quais serão vinculados dinamicamente
  - A checagem de tipo em Smalltalk é dinâmico, o que é flexível mas inseguro
  - Smalltalk é 10 vezes lento que C++