Classes e objetos II

📕 Funções inline

Em C++ é possível criar macros para realização de cálculos que se assemelham à chamada de funções [1]. Por exemplo:

```
#define quadrado(x) ((x) * (x))
```

A vantagem do uso de macros é que elas não criam registros de ativação quando são chamadas. Por outro lado, elas não possuem validação de tipos em tempo de compilação e possuem problemas de legibilidade para expressões muito complexas [1].

O recurso de funções inline funciona de maneira semelhante ao das macros e não geram registros de ativação durante a sua chamada. Além disso, possuem a sintaxe igual ao da declaração de funções e validação de tipos em tempo de compilação [1].

Por não gerarem código compilado, toda vez que uma chamada de função inline é realizada, o compilador precisa compilar a função para incluir no arquivo objeto. Isso pode aumentar o tempo durante a compilação do projeto. Por isso, as funções inline são indicadas para operações simples realizadas em poucas linhas de código. A declaração de uma função inline é feita usando a palavra reservada `inline` [1].

```
inline int quadrado(int x) {
    return x * x;
}
```

Para que uma função inline seja acessível em mais de um módulo de um projeto, é necessário que ela esteja declarada completa no arquivo de cabeçalho [1].

📙 Métodos inline

A declaração de um método inline pode ser realizada de duas maneiras [1]:

- Na declaração da classe:
  - Não precisa da palavra-chave `inline`
- Fora da declaração da classe:
  - Precisa da palavra-chave `inline`
  - Implementação precisa estar no arquivo cabeçalho

```
// arquivo quadrado.h

class Quadrado {
public:
    // Método inline A
    Quadrado(int lado) {
        _lado = lado
    }

    int calcularArea();

private:
    int _lado;
};

// Método inline B
inline int Quadrado::calcularArea() {
    return _lado * _lado;
}
```

📗 Objetos constantes

Em C++, o comando `const` é usado para declaração de constantes. Uma constante não pode ter seu valor alterado depois de feita a sua atribuição inicial. A ideia de se declarar constantes é diferenciar as variáveis que podem ou não ter seus valores alterados. Desta maneira, se por engano alguém tentar alterar uma variável, que não deveria ser modificada, um erro é gerado em tempo de compilação [1].

```
const int a = 10;
a = 20; // ERRO!
```

Para declaração de objetos constantes, o comando `const` é utilizado da mesma maneira [1]. Por exemplo:

```
const Quadrado quadrado(2);
quadrado.calcularArea(); // ERRO!
```

Um objeto constante não pode ter o valor de nenhum de seus atributos alterados. Com isso, os métodos mutantes não podem ser chamados de um objeto constate. Para chamada dos métodos que não modificam o estado interno do objeto é necessário indicar na declaração da classe que o método é constante [1]. Por exemplo:

```
class Aluno {
public:

    Aluno(string nome) : _nome(nome){}

    void setNome(string nome) {
        _nome = nome
    }

    // const indica que o método não modifica o objeto
    string getNome() const {
        return _nome;
    }

    void imprime() const {
        cout << _nome << endl;
    }
private:
    string _nome;
};
```

📘 Membros estáticos

Em C++, é possível declarar uma variável que é compartilhada por todos os objetos de uma classe. Caso um objeto altere o valor da variável, todos os objetos vão acessar o mesmo valor alterado. A palavra-chave `static` é usada para a declaração de variáveis de classe [1].

Na prática este tipo de variável não é utilizada pois possuem o mesmo comportamento de variáveis globais. Isto pode ocasionar alguns problemas inesperados e de difícil depuração. No entanto, as variáveis do tipo `static` podem ser utilizadas para declaração de constantes no escopo de uma classe. Assim, a constante não precisa ser copiada para todos os objetos e pode ser compartilhada sem riscos de efeitos colaterais [1].

```
class Eleitor {
public:
    static int numEleitores;

    Eleitor() {
        Eleitor::numEleitores++;
    }

private:
    // Declaração de constante interna
    const static int MAX_ELEITORES = 10;
};

// Inicializa o membro estático
int Eleitor::numEleitores = 0;

int main() {
    Eleitor fred;
    Eleitor velma;
    Eleitor scooby;
    cout << Eleitor::numEleitores << endl;
}
```

👨‍💻 Atividade prática

1. Altere a classe Relógio utilizando apenas métodos inline, quando aplicável.
2. Altere a classe Relógio indicando quais são os métodos constantes.
3. Altere a classe Relógio adicionando duas constantes: `HORA_MAX = 23` e `MINUTO_MAX = 59`. Altere o código para utilizar estas constantes na validação dos métodos.

📝 Atividade teórica

1. Quais são as vantagens de se utilizar funções `inline`?
2. Em quais cenários as funções inline devem ser evitadas?
3. O que são objetos constantes e quais são suas aplicações?
4. Podemos sempre criar nossas classes sem levar em consideração a indicação do `const` nos métodos? Justifique.
5. O que são membros estáticos de uma classe? Quais suas aplicações?

📖 Leitura recomendada

- Capítulo 8: MIZRAHI, Vctorine Viviane. **Treinamento em Linguagem C++ - Módulo II**. Makron Books,1994.

📚 Referência bibliográficas

[1] MIZRAHI, Vctorine Viviane. **Treinamento em Linguagem C++ - Módulo II**. Makron Books,1994.