A04 Classes e objetos

📕 Passagem de parâmetros por referência

Em C++, toda variável passada como parâmetro para uma função ou métodos é sempre passada como **valor**. Isso significa que, em tempo de execução, o valor da variável é copiado para a variável de parâmetro. Se o parâmetro for modificado, a variável de chamada permanece inalterada[1].

```
void incrementa(int a) {
    a++;
}

int main() {
    int x = 10;
    cout << x << endl; // 10
    incrementa(x);
    cout << x << endl; // 10
}
```

O mesmo comportamento permanece quando se trabalha com a passagem de objetos para funções e métodos. Quando um objeto é passado por parâmetro, todos os seus atributos são copiados para a variável de argumento da função[1].

```
void calculaNota(Aluno a) { // a = cópia do objeto
    a.setNota(10);
}

int main() {
    Aluno a; // nota = 0
    calculaNota(a);
    cout << a.getNota() << ends; // nota = 0
}
```

Uma maneira de permitir que uma variável seja alterada dentro de uma função é realizar a passagem dos parâmetros usando ponteiros[1].

```
void calculaNota(Aluno *a) { // a = endereço do objeto
    a->setNota(10);
}

int main() {
    Aluno a; // nota = 0
    calculaNota(&a);
    cout << a.getNota() << ends; // nota = 10
}
```

Com a utilização de ponteiros, o que é passado como parâmetro é o endereço da variável (8 bytes). Uma das desvantagens do uso de ponteiros é que ela possibilita que erros passem desapercebidos em tempo de compilação e ocorram apenas durante a execução do programa[1]. Alguns erros:

- `calculaNota(a);`: esquecer de colocar o operador `&` indicando o endereço da variável.
- `a.setNota(10);`: utilizar o operador `.` no lugar do operador `->`.

Uma maneira de simplificar isso é utilizando o tipo referência.

📙 Tipo referência

O tipo referência é um tipo semelhante ao ponteiro mas com algumas restrições. Uma variável do tipo referência armazena o endereço de outra variável, assim como um ponteiro. No entanto, após ser iniciada, uma variável do tipo referência não pode apontar para outra variável. Também não é possível utilizar operações de incremento e decremento de ponteiros em variáveis de referência.

```
int a = 100;
int &x = a;
```

```
// Alterando o valor de x, está altera
x = 20;
cout << a << endl; // 20
```

> 🚨 **Atenção** 🚨
>
> Um tipo referência só pode ser inicializado com outra variável. Não é possível inicializar uma variável de referência com um valor literal
>
> Exemplo: `int &x = 10; // ERRO!`

A utilização do tipo referência pode ser utilizada com objetos e se comporta como se fosse um objeto normal, sem a necessidade do uso do operador `->`[1].

```
void calculaNota(Aluno &a) { // a = referência do objeto
    a.setNota(10);
}

int main() {
    Aluno a; // nota = 0
    calculaNota(a);
    cout << a.getNota() << ends; // nota = 10
}
```

Quando se trabalha com objetos, o ideal é que eles sempre sejam passados como referência. Isso porque ao passar o objeto por cópia todos seus atributos acabam sendo copiados para a variável de parâmetro da função. Se o objeto possuir muitos atributos, ou um vetor estático muito grande, a passagem por cópia vai prejudicar o desempenho do programa[1].

Por via de regra, objetos devem sempre ser passados por referência. Nos casos onde não se deseja que o objeto seja alterado dentro da função, o parâmetro precisa ser declarado como `const`[1].

Sumarizando, com os tipos referência temos os seguintes benefícios:

- Eliminação dos erros no trabalho com ponteiros: esquecer o `&` na chamada do método ou o `->` nas referências do objeto.
- Somente o endereço da variável é copiado: evita a sobrecarga de copiar o objeto inteiro.
- Evitar que o objeto seja modificado dentro de um método: os parâmetros `const` asseguram que o objeto será imutável.

👨‍💻 Atividade prática

1. Crie uma classe `Cliente` com os seguintes atributos: nome, email, data de nascimento e idade. Crie os métodos acessores e mutantes indicando corretamente se os métodos são constantes ou não. Utilize o tipo referência para todos os parâmetros objetos.
2. Crie um classe `Conta` com os seguintes atributos: número, agência, Cliente e saldo. Crie um construtor que inicializa todos os atributos. Crie um método para depositar uma quantia, um método para retirar uma quantia, um método para imprimir o extrato da conta e outro método para alterar o cliente da conta.
3. Crie uma classe `Hora` que representa um horário com hora, minutos e segundos. Implemente as operações de soma e subtração de horas. Indique corretamente os métodos que devem ser constantes, os parâmetros constantes e do tipo referência. Exemplo de uso da operação soma: `hora1.soma(hora2);`

📝 Atividade teórica

1. Descreva as vantagens da utilização dos tipos referência em C++.
2. Quais são os principais usos dos tipos referência?
3. Uma função pode retornar um valor do tipo referência? Justifique.

4. Analise as três funções a seguir:

```
int funcaoA(ObjetoGrande x)  { ... }
int funcaoB(ObjetoGrande &x) { ... }
int funcaoC(ObjetoGrande *x) { ... }
```

   a. Qual dessas funções executa mais rápido?
   b. Qual dessas funções tem execução mais lenta?

📖 Leitura recomendada

- Capítulo 11: 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. ↩