"É o hardware que torna um computador rápido. É o software que transforma um computador rápido em lento" (Craig Bruce).

Referências e o Ponteiro this

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Referências

Para evitar a cópia completa de um objeto, podemos passar apenas o seu ponteiro.

Ainda melhor do que passar um ponteiro, podemos passar uma **referência**.

Referências

Assim como um ponteiro, uma referência “aponta para a variável original”.

Referências

Assim como um ponteiro, uma referência “aponta para a variável original”.

Diferente de um ponteiro ...

- Você pode utilizar uma referência como se fosse um objeto (variável) “normal”.
- Referências garantidamente apontam para algum objeto.
- **Não existe referência nula.**
  - Mas você sempre pode dar um tiro no pé e criar uma referência para algo que é deletado da memória em algum momento.

Exemplo

Criar uma função membro em Disciplina que imprime os dados da disciplina.

Antes da impressão dos dados, um cabeçalho deve ser impresso.

Precisamos imprimir quantos % da carga horária total do curso a disciplina representa.

Passar a carga horária total do curso e o cabeçalho como parâmetros para a função membro.

Passagem por cópia

```
#ifndef DISCIPLINA_H
#define DISCIPLINA_H

#include <string>

#include "Pessoa.hpp"

class Disciplina{

      //...
       void imprimirDados(
            std::string cabecalho,
            unsigned int cargaTotalCurso);

   private:
       std::string nome;
       unsigned short int cargaHoraria;
       Pessoa* professor;
};
#endif
```

```
#include "Disciplina.hpp"

#include <iostream>

//...

void Disciplina::imprimirDados(std::string cabecalho, unsigned int
cargaTotalCurso){
   double pctCurso = (double)cargaHoraria/cargaTotalCurso;
   pctCurso = pctCurso * 100;
   std::cout << cabecalho << std::endl;
   std::cout << "Disciplina: " << nome << std::endl;
   std::cout << "Carga: " << cargaHoraria << std::endl;
   std::cout << "Porcentagem do curso: " << pctCurso << "%" << std::endl;
   std::cout << "Professor: " << professor->getNome() << std::endl;

}
```

```
int main(){
   Pessoa* p1{new Pessoa{"Joao", 11111111111, 20}};

   Disciplina d1{"Orientacao a Objetos"};
   d1.setProfessor(p1);
   d1.setCargaHoraria(60);

   std::string cabecalho{"Dados da disciplina"};
   d1.imprimirDados(cabecalho, 4000);

   delete p1;

   return 0;
}
```

Relembrando

Quais os problemas da passagem por cópia?

Relembrando

Quais os problemas da passagem por cópia?

Temos um **clone** do objeto original.

Alterações no objeto original não refletem em mudanças no clone, e vice-versa.

Mais memória ocupada.

A cópia do objeto custa caro, especialmente para objetos grandes e complexos.

No exemplo, estamos copiando o inteiro, e o objeto do tipo string.

Referência

```hpp
#ifndef DISCIPLINA_H
#define DISCIPLINA_H

#include <string>

#include "Pessoa.hpp"

class Disciplina{

      //...
       void imprimirDados(
            std::string& cabecalho,
            unsigned int cargaTotalCurso);

   private:
       std::string nome;
       unsigned short int cargaHoraria;
       Pessoa* professor;
};
#endif
```

```cpp
#include "Disciplina.hpp"

#include <iostream>

//...

void Disciplina::imprimirDados(std::string& cabecalho,
      unsigned int cargaTotalCurso){
   double pctCurso = (double)cargaHoraria/cargaTotalCurso;
   pctCurso = pctCurso * 100;
   std::cout << cabecalho << std::endl;
   std::cout << "Disciplina: " << nome << std::endl;
   std::cout << "Carga: " << cargaHoraria << std::endl;
   std::cout << "Porcentagem do curso: " << pctCurso << "%" << std::endl;
   std::cout << "Professor: " << professor->getNome() << std::endl;

}
```

Únicas alterações necessárias!

```cpp
int main(){
   Pessoa* p1{new Pessoa{"Joao", 11111111111, 20}};

   Disciplina d1{"Orientacao a Objetos"};
   d1.setProfessor(p1);
   d1.setCargaHoraria(60);

   std::string cabecalho{"Dados da disciplina"};
   d1.imprimirDados(cabecalho, 4000);

   delete p1;

   return 0;
}
```

Modificando para referências

Basta colocar um **&** antes do nome dos parâmetros para indicar que se trata de uma **referência**.

Temos um efeito parecido ao da passagem por ponteiros.

Ou até melhor, já que o compilador pode fazer otimizações extras.

Por exemplo, geralmente não são necessárias derreferências em tempo de execução para acessar a memória.

O compilador muitas vezes sabe onde a variável está na memória, e injeta diretamente o endereço acessar.

O código da função permanece como se estivéssemos utilizando um objeto convencional.

No main, nada precisou ser alterado.

Outro exemplo

```
int valor = 10;
int& refValor{valor};//refValor é uma referência para valor
std::cout << refValor << std::endl;
```

Regras e Boas Práticas

Toda referência **deve** ser inicializada no momento de sua criação ou na lista de inicialização de membros da classe.

Exemplo:

```
int valor = 10;
int& ref1; //erro de compilação
std::cout << ref1 << std::endl
```

Regras e Boas Práticas

Depois de atribuída, uma referência não pode apontar para um objeto diferente.

Exemplo:

```
int valor1 = 10;
int valor2 = 20;
int& refValor{valor1};//refValor é uma referência para valor
refValor{valor2};
//erro de compilação, depois de inicializado refValor não pode apontar para outra variável
```

Regras e Boas Práticas

Sempre que fizer sentido, utilize referências ao invés de ponteiros.

Mas tenha em mente que uma referência é **menos flexível do que um ponteiro.**

O Ponteiro this

Edite o makefile.

Adicione o parâmetro de compilação `-Wshadow`.

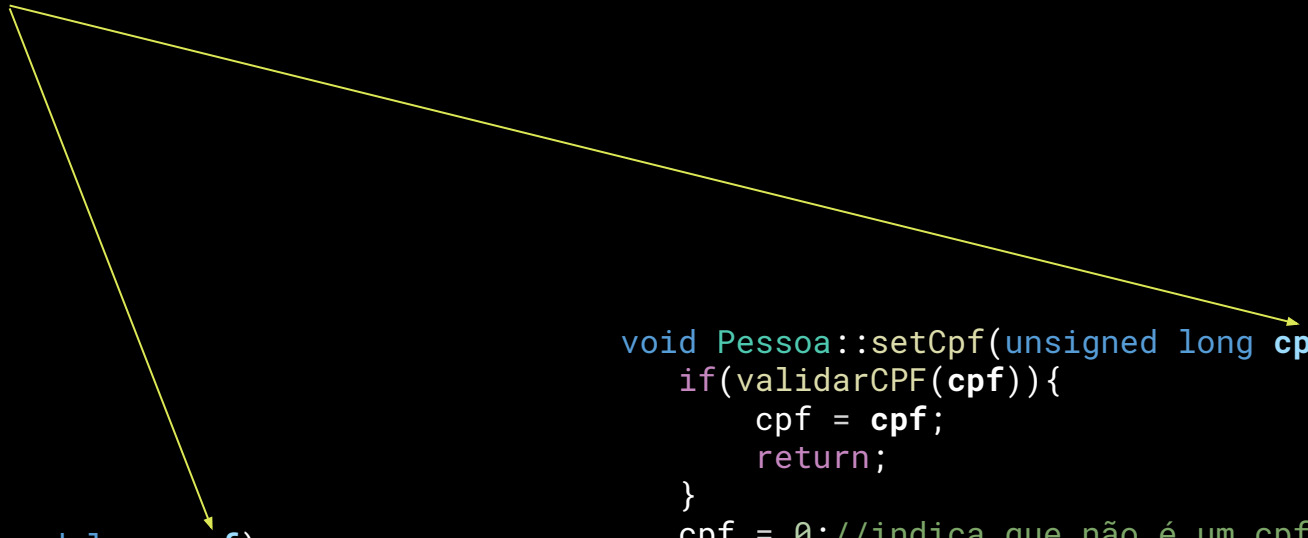
```
parametrosCompilacao=-Wall -Wshadow
```

A opção `-Wshadow` avisa se criamos alguma variável ou função que pode ocultar outra variável ou função.

Para detalhes, digite no terminal `man g++`, e procure por `-Wshadow`.

Altere

Altere o nome do parâmetro em Pessoa.



```
void setCpf(unsigned long cpf);
```

```
void Pessoa::setCpf(unsigned long cpf){
    if(validarCPF(cpf)){
        cpf = cpf;
        return;
    }
    cpf = 0;//indica que não é um cpf válido
    return;
}
```

Algumas Perguntas

Algumas perguntas.

- Podemos fazer isso?

```
void setCpf(unsigned long cpf);
```

```
void Pessoa::setCpf(unsigned long cpf){
   if(validarCPF(cpf)){
       cpf = cpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}
```

Algumas Perguntas

Algumas perguntas.

- Podemos fazer isso?
  - Sim, podemos. Mas note que existe um dado membro chamado cpf.
  - Como cpf foi declarado dentro da função novamente, o dado membro (cpf) será ignorado, e somente a variável local cpf será referenciada.
    - O dado membro cpf está sendo sombreado (shadowed) pela variável local cpf
- Funciona?

```
void setCpf(unsigned long cpf);
```

```
void Pessoa::setCpf(unsigned long cpf){
   if(validarCPF(cpf)){
       cpf = cpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}
```

Algumas Perguntas

Algumas perguntas.

- Podemos fazer isso?
  - Sim, podemos. Mas note que existe um dado membro chamado cpf.
  - Como cpf foi declarado dentro da função novamente, o dado membro (cpf) será ignorado, e somente a variável local cpf será referenciada.
    - O dado membro cpf está sendo sombreado (shadowed) pela variável local cpf
- Funciona?
  - Do jeito que fizemos não.
    - A variável local recebe o valor dela mesma!?

```
void setCpf(unsigned long cpf);
```

```
void Pessoa::setCpf(unsigned long cpf){
    if(validarCPF(cpf)){
        cpf = cpf;
        return;
    }
    cpf = 0;//indica que não é um cpf válido
    return;
}
```

Faça você mesmo

Limpe os arquivos objeto e compile novamente

```
make clean
make
```

Veja que o `-Wshadow` nos avisou da besteira que fizemos.

```
... warning: declaration of ‘cpf’ shadows a member of ‘Pessoa’ [-Wshadow]
 void Pessoa::setCpf(unsigned long cpf){
                                   ^
...
Pessoa.hpp:28:17: note: shadowed declaration is here
   unsigned long cpf;
                 ^~~
...
```

Como resolver

É necessária uma forma de dizer ao compilador a “qual cpf” estamos nos referenciando.

```
void Pessoa::setCpf(unsigned long cpf){
   if(validarCPF(cpf)){
       cpf = cpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}
```

Aqui é o dado membro cpf

Aqui é o parâmetro cpf

O Ponteiro this

O ponteiro especial `this` **aponta para o objeto atual na memória**.

Contém o endereço do objeto atual.

O ponteiro existe implicitamente para todos os objetos.

Todo objeto tem seu próprio `this` (você não precisa declarar).

Faça você mesmo

Crie uma função membro em `Pessoa` que imprime o endereço de memória do objeto `Pessoa` instanciado atualmente.

```
void Pessoa::imprimirEnderecoMemoria();
```

```
void Pessoa::imprimirEnderecoMemoria(){
   std::cout << this << std::endl;
}
```

```
#include<iostream>

#include "Pessoa.hpp"

int main(){
   Pessoa* p1{new Pessoa{"Joao", 11111111111, 20}};
   Pessoa p2{"Maria"};

   p1->imprimirEnderecoMemoria();
   p2.imprimirEnderecoMemoria();

   delete p1;

   return 0;
}
```

O Ponteiro this

Como utilizar o this para resolver o problema na função membro setCpf?

```
void Pessoa::setCpf(unsigned long cpf){
   if(validarCPF(cpf)){
       cpf = cpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}
```

O Ponteiro this

Como utilizar o this para resolver o problema na função membro setCpf?

Basta adicionar this na frente do cpf que desejamos que aponte para o dado membro.

Lembre-seque para acessar um membro via ponteiro utilizamos o operador seta ->.

```
class Pessoa{
   public:

     //…

   private:
       bool validarCPF(unsigned long cpfTeste);

       std::string nome;
       unsigned long cpf;
       unsigned char idade;
};
```

Se refere ao dado membro cpf

```
void Pessoa::setCpf(unsigned long cpf){
   if(validarCPF(cpf)){
       this->cpf = cpf;
       return;
   }
   this->cpf = 0;//indica que não é um
cpf válido
   return;
}
```

Dica

Remova o `-Wshadow` dos parâmetros de compilação.

Caso contrário o compilador vai continuar reclamando devido ao nome do parâmetro ser o mesmo do dado membro.

Já resolvemos o problema através do this->.

Boas práticas de programação

Nos sets e construtores utilize como nomes de parâmetros **o mesmo nome dos dados membro**.

Evitar a proliferação de identificadores.

Use o ponteiro this para identificar quando você está se referindo aos dados membro.

Sempre use o this nas funções membro quando se referenciar a um dado membro.

Mesmo que não haja uma variável fazendo sombreamento.

Dica

O ponteiro `this` existe em diversas linguagens, e seu uso é amplamente encorajado.

Alguns exemplos (Note que nessas linguagens não existem ponteiros, mas o uso do this é similar).

this em Java: www.w3schools.com/java/ref_keyword_this.asp

this em C#: docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/this

self em Python: www.geeksforgeeks.org/self-in-python-class

Exercícios

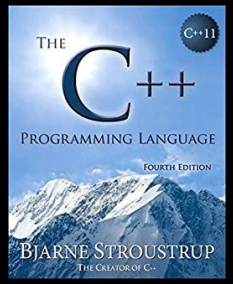
1. Crie uma classe Curso para representar cursos (ex.: BCC, IBM, ...). A classe deve conter dados membro como nomeCurso, anoCriacao, cargaHorariaMinima, ... Adicione um membro dentro de Disciplina, que representa o curso ao qual a disciplina está vinculada. Dentro de disciplina, o curso deve ser uma referência a um objeto. Para que isso funcione, todos os construtores de Disciplina devem receber o curso a que a disciplina pertence como parâmetro (lembre-se que não existe referência não inicializada). Note também que uma disciplina não poderá mudar de curso, já que uma referência depois de inicializada, não pode "apontar" para outro objeto.

2. Modifique todos os sets e construtores das classes para que os parâmetros possuam o mesmo nome dos dados membro. Na lista de inicializador de membro dos construtores não utilize o this.O compilador já sabe que o conteúdo que está entre chaves se refere ao parâmetro, e o que está fora das chaves é o dado membro. Exemplo:

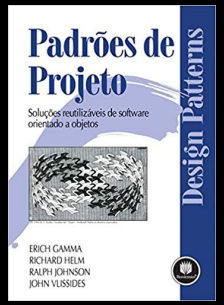
```
Pessoa::Pessoa(std::string nome):nome{nome} { }
```

Referências

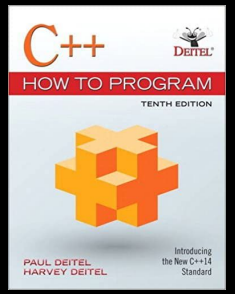
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



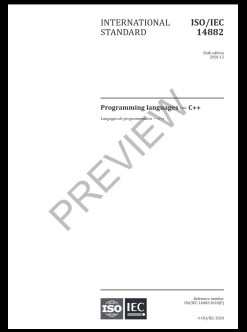
Gamma, E. Padrões de Projetos: Soluções Reutilizáveis. Bookman. 2009.



Deitel, H. M., Deitel, P. J. C++: como programar. 5a ed. Pearson Prentice Hall. 2006.



ISO/IEC 14882:2020 Programming languages - C++: www.iso.org/obp/ui/#iso:std:iso-iec:14882:ed-6:v1:en



Licença