Referências

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

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 para as funções

Referências

- Assim como um ponteiro, uma referência “aponta para a variável original”
- Principais diferenças
  - Você pode utilizar uma referência como se fosse um objeto “normal”
  - Referências garantidamente apontam para algum objeto
    - Não existe referência nula
      - Você sempre pode dar um tiro no pé e criar uma referência para algo que é deletado da memória em algum momento
        - Para evitar esses problemas, basta não fazer besteira!!!

Exemplo

- Criar um método em Disciplina que imprime os dados da disciplina
  - Além disso
    - Antes da impressão dos dados, um cabeçalho deve ser impresso
    - Precisamos imprimir quantos % da carga horária total do curso a disciplina representa
    - Vamos precisar passar a carga horária total do curso e o cabeçalho como parâmetros para a função membro

Exemplo – Usando passagem por cópia

Em Disciplina.hpp

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

Em Disciplina.cpp

```
//...
void Disciplina::imprimeDados(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 << "Pct do curso: " << pctCurso << "%" << std::endl;
        std::cout << "Professor: " << professor->getNome() << std::endl;
}
```

No main.cpp

```
//...
int main(){
        Pessoa* p1{new Pessoa{"Joao", 20}};
        Disciplina disciplina{"C++"};
        disciplina.setProfessor(p1);
        disciplina.setCargaHoraria(72);


        std::string cabecalho = "Dados da disciplina";
        unsigned int cargaHorariaTotal = 3000;
        disciplina.imprimeDados(cabecalho,
                cargaHorariaTotal);


        delete p1;


        return 0;
}
```

UDESC

Relembrando

- Relembrando das últimas aulas, qual o problema da passagem por cópia?

Relembrando

- Relembrando das últimas aulas, qual o problema da passagem por cópia?
  - Temos um **clone** do objeto original
    - Alterações no objeto original não refletem em mudanças no clone
  - Ocupamos mais memória
  - A cópia do objeto custa caro, especialmente para objetos grandes e complexos
- No exemplo, estamos copiando o inteiro, e o objeto **string**

Modificando para referências

Em Disciplina.hpp

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

Em Disciplina.cpp

```
//...
void Disciplina::imprimeDados(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 << "Pct do curso: " << pctCurso << "%" << std::endl;
        std::cout << "Professor: " << professor->getNome() << std::endl;
}
```

No main.cpp – **Nada alterado!**

```
//...
int main(){
        Pessoa* p1{new Pessoa{"Joao", 20}};
        Disciplina disciplina{"C++"};
        disciplina.setProfessor(p1);
        disciplina.setCargaHoraria(72);

        std::string cabecalho = "Dados da disciplina";
        unsigned int cargaHorariaTotal = 3000;
        disciplina.imprimeDados(cabecalho,
                cargaHorariaTotal);

        delete p1;

        return 0;
}
```

Modificando para referências

- Basta colocar um & antes do nome dos parâmetros para indicar que se trata de uma referência
  - Temos o mesmo efeito do que se fizéssemos a passagem por ponteiros
  - Mas note que o “ponteiro” fica escondido
    - O código da função permanece como se estivéssemos utilizando um objeto convencional
    - No main, não precisamos mudar nada
      - Não precisamos passar o endereço

Outros exemplos

```
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
  - Não são permitidas referências não inicializadas
    - Exemplo:

```
int valor = 10;
int& refValor{valor};//refValor é uma referência para valor
int& ref2; //erro de compilação
std::cout << refValor << std::endl;
```

Regras e boas práticas

- Depois de atribuída, uma referência não pode ser trocada
  - Uma referência começa “apontando” para um objeto, e o objeto apontado **não pode ser trocado**
    - 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

Exercício

- Crie uma classe chamada curso
  - Representa um curso (ex.: BCC), e contém dados membro como nomeCurso, anoCriacao, cargaHorariaMinima ...
  - Adicione um membro dentro de Disciplina, que representa o curso ao qual a disciplina está vinculada
  - O Curso deve ser uma **referência**
    - 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

Referências

- DEITEL, P.; DEITEL, H. **C++ how to Program**. [S.l.]: Pearson, 2017. ISBN 9780134448237
- STROUSTRUP, B. **The C++ Programming Language**. Pearson Education, 2013. ISBN 9780133522853.