const

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

const

- Ao declarar um objeto ou variável como ***const***, estamos indicando que o objeto (ou variável) não pode ser alterado depois de inicializado
  - Tentar alterar um objeto *const* resulta em um **erro de compilação**
  - Exemplos

```
#include <iostream>

#include "Pessoa.hpp"

int main(){
	const Pessoa p{"Joao", 20, 11111111111};
	const int valor{1};

	//...

	return 0;
}
```

const

- Ao declarar um objeto ou variável como ***const***, estamos indicando que o objeto (ou variável) não pode ser alterado depois de inicializado
  - Tentar alterar um objeto *const* resulta em um **erro de compilação**
  - Exemplos

```
#include <iostream>

#include "Pessoa.hpp"

int main(){
        const Pessoa p{"Joao", 20, 11111111111};
        const int valor{1};

        valor++;

        //...

        return 0;
}
```

Erro de compilação

Acessando funções membro

- Você não pode acessar membros de um objeto *const*
  - Somente membros *const* podem ser acessados
    - Caso contrário, um set por exemplo poderia alterar o objeto (que foi declarado como *const*)

```
#include "Pessoa.hpp"

int main(){
	const Pessoa p{"Joao", 20, 11111111111};
	const int valor{1};

	std::cout << p.getNome() << std::endl;
	std::cout << valor << std::endl;

	return 0;
}
```

Erro de compilação

Funções const

- Ao declarar uma função membro como *const*, estamos dando garantias ao compilador de que a função não modifica o objeto
  - Não modifica os dados membro do objeto
- Que tipos de função tipicamente devem ser *const*?

Funções const

- Ao declarar uma função membro como *const*, estamos dando garantias ao compilador de que a função não modifica o objeto
  - Não modifica os dados membro do objeto
- Que tipos de função tipicamente devem ser *const*?
  - Os gets são const, já que não alteram o objeto
    - Comumente **retornam uma cópia** de algum dado membro

Exemplo

- Faça as alterações

Para indicar que a função é **const**

Pessoa.hpp

```
//...
std::string getNome() const;
//...
```

Pessoa.cpp

```
//...
std::string Pessoa::getNome() const{
    return this->nome;
}
//...
```

UDESC

Teste você mesmo

main.cpp

Agora podemos chamar essa função

```
#include <iostream>

#include "Pessoa.hpp"

int main(){
	const Pessoa p{"Joao", 20, 11111111111};
	const int valor{1};

	std::cout << p.getNome() << std::endl;
	std::cout << valor << std::endl;

	return 0;
}
```

Não esqueça de fazer um *make clean* antes de compilar!!!

UDESC

Teste você mesmo

- Tente fazer com que a função setNome seja const
  - Modifique no .hpp e no .cpp
- *make clean*
- *make*
- O que acontece? Por que?

Teste você mesmo

- Tente fazer com que a função setNome seja const
  - Modifique no .hpp e no .cpp
- *make clean*
- *make*
- O que acontece? Por que?
  - Temos um erro de compilação
  - O compilador detectou que você fez uma promessa e não a cumpriu
    - A função setNome modifica o estado do objeto, logo não pode ser *const*
      - Modifica o nome da pessoa

Observações

- Construtores e destrutores **não podem** ser *const*
- Construtores e destrutores podem modificar os itens *const* da classe
  - Os itens *const* da classe são constantes apenas depois do construtor terminar de construir o objeto, e até a chamada do destrutor

Parâmetros const

- Podemos indicar que os parâmetros de nossas funções são *const*
  - Indica que a função vai apenas ler os dados desse parâmetro, e não vai alterá-lo de forma alguma

Parâmetros const

- Podemos indicar que os parâmetros de nossas funções são *const*
  - Indica que a função vai apenas ler os dados desse parâmetro, e não vai alterá-lo de forma alguma
  - Especialmente útil quando passamos parâmetros via referência
    - Por quê?

Parâmetros const

- Podemos indicar que os parâmetros de nossas funções são *const*
  - Indica que a função vai apenas ler os dados desse parâmetro, e não vai alterá-lo de forma alguma
  - Especialmente útil quando passamos parâmetros via referência
    - Uma passagem por referência é mais leve para objetos complexos, mas como garantir que a função chamada não vai alterar o nosso dado original?
    - Se o parâmetro é *const*, essa garantia é dada

Exemplo

- Modifique a função setNome de pessoa para receber uma referência para a string do nome
  - Evitamos que a string toda seja copiada ao ser passada para a função
    + rápido
  - Vamos garantir que a string original não é alterada, anotando-a como *const*

Pessoa.hpp

```
void setNome(const std::string& nome);
```

Pessoa.cpp

```
void Pessoa::setNome(const std::string& nome) {
    this->nome = nome;
}
```

UDESC

Dica de performance

- Variáveis *const* podem abrir espaço para muitas otimizações de performance
- O compilador pode realizar otimizações que não são possíveis em variáveis não *const*
  - O compilador pode, por exemplo, salvar os dados no segmento de dados ou segmento de texto do programa
    - Os cálculos de endereço são mais simples nesses segmentos

Teste você mesmo

```
#include<iostream>


int main(){
    int valor{1330};

    std::cout << valor << std::endl;

    return 0;
}
```

Assembly x86 gerado no g++ 7.5.0

g++ -S main.cpp

```
...
main:
.LFB1493:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $1330, -4(%rbp)
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        leaq    _ZSt4cout(%rip), %rdi
        call    _ZNSolsEi@PLT
        ...
```

UDESC

Teste você mesmo

```
#include<iostream>


int main(){
    const int valor{1330};

    std::cout << valor << std::endl;

    return 0;
}
```

Assembly x86 gerado no g++ 7.5.0

g++ -S main.cpp

```
...
main:
.LFB1493:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $1330, -4(%rbp)
        movl    $1330, %esi
        leaq    _ZSt4cout(%rip), %rdi
        call    _ZNSolsEi@PLT
        movq    %rax, %rdx
        ...
```

UDESC

Teste você mesmo

Sem const

```
...
main:
.LFB1493:
        .cfi_startproc
        pushq       %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq        %rsp, %rbp
        .cfi_def_cfa_register 6
        subq $16, %rsp
        movl $1330, -4(%rbp)
        movl -4(%rbp), %eax
        movl %eax, %esi
        leaq  _ZSt4cout(%rip), %rdi
        call   _ZNSolsEi@PLT
        ...
```

O compilador não precisou carregar o valor da memória, pois assumiu que sempre vai ser 1330 → e carregou como uma constante (valor imediato) para o registrador esi

Com const

```
…
main:
.LFB1493:
        .cfi_startproc
        pushq       %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq        %rsp, %rbp
        .cfi_def_cfa_register 6
        subq $16, %rsp
        movl $1330, -4(%rbp)
        movl $1330, %esi
        leaq  _ZSt4cout(%rip), %rdi
        call   _ZNSolsEi@PLT
        ...
```

UDESC

Const ou não const, eis a questão

- O parâmetro idade deveria ser *const*? Há algum ganho nisso?

```
void Pessoa::setIdade(unsigned short int idade){
    this->idade =  idade;
}
```

Const ou não const, eis a questão

- O parâmetro idade deveria ser *const*? Há algum ganho nisso?
  - A passagem é feita por cópia, e não ponteiro ou referência
  - O valor original da variável passada como parâmetro não é alterado
    - Como *const* garante que o valor original não vai ser alterado, isso se torna redundante
  - No entanto ainda podemos ter ganhos
    - Do ponto de vista da engenharia de software
      - Deixamos claro que esse parâmetro é algo que vai ser usado somente para leitura
    - Do ponto de vista da performance
      - O compilador pode otimizar as coisas, como discutido anteriormente

```
void Pessoa::setIdade(unsigned short int idade){
    this->idade =  idade;
}
```

UDESC

Principle of Least Privilege

- **Princípio do menor privilégio**
  - Suas funções devem ter **apenas acesso o suficiente** aos dados para poder cumprir sua tarefa, **e não mais do que isso** (DEITEL; DEITEL, 2017)

Próxima aula

- O modificador const se torna ainda mais importante quando usado com ponteiros e retornos
  - Veremos na próxima aula

Exercício

1. Modifique todas as funções e parâmetros de função para const quando isso fizer sentido
   - Por enquanto **ignore** funções que envolvem **ponteiros**

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.