"A vida é prazerosa. A morte é pacífica. É a transição que é problemática" (Isaac Asimov).

Asserts

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Considere...

```cpp
#include <iostream>
#include <cmath>

int main(){
   int v1, v2;
   std::cin >> v1 >> v2;
   int resultado {std::abs(v1) + std::abs(v2)};

   std::cout << resultado << '\n';

   return 0;
}
```

Considere...

```cpp
#include <iostream>
#include <cmath>

int main(){
   int v1, v2;
   std::cin >> v1 >> v2;
   int resultado {std::abs(v1) + std::abs(v2)};

   std::cout << resultado << '\n';

   return 0;
}
```

O resultado é garantidamente positivo.

Exceto ... :

- Se a função abs possuir algum erro.
- Se tivermos um overflow.

Considere...

```cpp
#include <iostream>
#include <cmath>

int main(){
   int v1, v2;
   std::cin >> v1 >> v2;
   int resultado {std::abs(v1) + std::abs(v2)};

   std::cout << resultado << '\n';

   return 0;
}
```

Poderíamos testar um com if.

Mas vai causar um overhead desnecessário para testar uma condição que só será verdade caso o software tenha algum bug!

Asserts

```
#include <iostream>
#include <cmath>
#include <cassert>

int main(){
   int v1, v2;
   std::cin >> v1 >> v2;
   int resultado {std::abs(v1) + std::abs(v2)};

   assert(resultado >= 0);

   std::cout << resultado << '\n';

   return 0;
}
```

Nesses casos, faça um `assert`.

Testa se determinada condição é verdadeira em **tempo de execução**.

Se for falsa, a execução é abortada e é exibido o erro, o arquivo de código fonte que o originou, e a linha onde estava o assert.

Útil para testar se a aplicação nunca gera **valores absurdos em ambiente de desenvolvimento**.

Asserts

```cpp
#include <iostream>
#include <cmath>
#include <cassert>

int main(){
   int v1, v2;
   std::cin >> v1 >> v2;
   int resultado {std::abs(v1) + std::abs(v2)};

   assert(resultado >= 0);

   std::cout << resultado << '\n';

   return 0;
}
```

```
a.out: main.cpp:10: int main():
Assertion `resultado >= 0' failed.
Abortado (imagem do núcleo gravada)
```

Removendo asserts

Em ambiente de produção, podemos remover os asserts (e o seu overhead) compilando com `-DNDEBUG`.

Exemplo: `g++ main.cpp -DNDEBUG`

Faça você mesmo

Faça o programa anterior, e verifique a diferença no **assembly** pedindo para o compilador considerar, e desconsiderar os *asserts*.

```
g++ main.cpp
```

```
g++ main.cpp -DNDEBUG
```

Comparando

`g++ main.cpp`

```
main:
    ...
    cmovns    %edx, %eax
    addl    %ecx, %eax
    movl    %eax, -12(%rbp)
    cmpl    $0, -12(%rbp)
    js    .L2
    leaq    .LC0(%rip), %rax
    movq    %rax, %rcx
    movl    $10, %edx
    leaq    .LC1(%rip), %rax
    movq    %rax, %rsi
    leaq    .LC2(%rip), %rax
    movq    %rax, %rdi
    call    __assert_fail@PLT
.L2:
    movl    -12(%rbp), %eax
    movl    %eax, %esi
    ...
```

`g++ main.cpp -DNDEBUG`

```
main:
    ...
    cmovns    %edx, %eax
    addl    %ecx, %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %eax
    movl    %eax, %esi
    ...
```

Considere

Considere agora que precisamos garantir que um inteiro possui 4 bytes para poder compilar determinado programa.

```
#include <iostream>
#include <cassert>

int main(){
   assert(sizeof(int) >= 4);

   //...
   std::cout << "Fim!\n";

   return 0;
}
```

Considere

Nesse caso é possível responder a essa pergunta em **tempo de compilação**, e usar um assert é **incorreto**.

```
#include <iostream>
#include <cassert>

int main(){
   assert(sizeof(int) >= 4);

   //...
   std::cout << "Fim!\n";

   return 0;
}
```

Considere

Para perguntas que podem ser respondidas em tempo de compilação, use `static_assert`.

Funciona de maneira similar ao assert, mas nesse caso o binário não sofre alteração (mesmo sem o -DNDEBUG).

Tudo é resolvido em tempo de compilação.

```
#include <iostream>
#include <cassert>

int main(){
   static_ssert(sizeof(int) >= 4);

   //...
   std::cout << "Fim!\n";

   return 0;
}
```

Faça você mesmo

Compila o programa anterior com e sem o static_assert, e verifique que o assembly gerado é o mesmo.

```
g++ main.cpp
```

Atenção

Asserts devem ser usados para **testes de sanidade**.

Garantir que condições absurdas não são geradas durante a execução do programa.

Asserts podem ser mantidos apenas em desenvolvimento (durante o *debug*), e removidos através de NDEBUG no build final do programa.

static_asserts podem ser mantidos, já que não geram alterações no binário compilado.

Mas podem deixar a compilação mais demorada.

Para tratar condições “excepcionais”, mas que podem acontecer durante a execução do programa, use exceções (veremos adiante na disciplina).

Dicas

Veja as diretivas de desenvolvimento do Google.

Note que não é indicado o uso de variáveis unsigned, exceto em casos específicos.

É indicado que variáveis que nunca deveriam ser negativas sejam verificadas através de um **assert**.

https://google.github.io/styleguide/cppguide.html#Integer_Types

Dicas

Veja as diretivas de desenvolvimento do Google.

Note que não é indicado o uso de variáveis unsigned, exceto em casos específicos.

É indicado que variáveis que nunca deveriam ser negativas sejam verificadas através de um **assert**.

https://google.github.io/styleguide/cppguide.html#Integer_Types

A preocupação do Google é com a aritmética modular dos tipos sem sinal, que muitos programadores acabam não percebendo. Por exemplo, o seguinte loop nunca terminará (você consegue entender o motivo?).

```
for(unsigned int i{10}; i >= 0; --i){
      std::cout << i << '\n';
  }
```

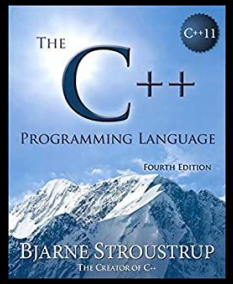
Dicas

O conceito de assert está disponível a partir do C++11, e foi melhorado no C++17.

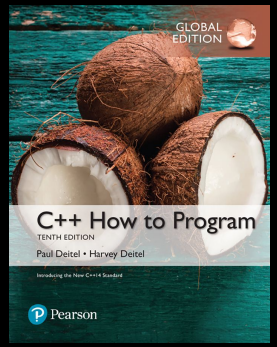
É a base para os **Concepts**, disponíveis a partir do C++20.

Referências

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

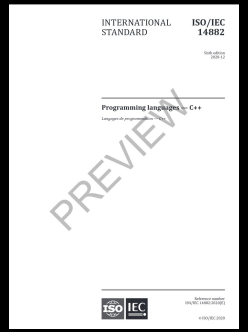


Deitel, H. M., Deitel, P. J. C++: como programar. 10a ed. Pearson Prentice Hall. 2017.



https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression.

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



Licença