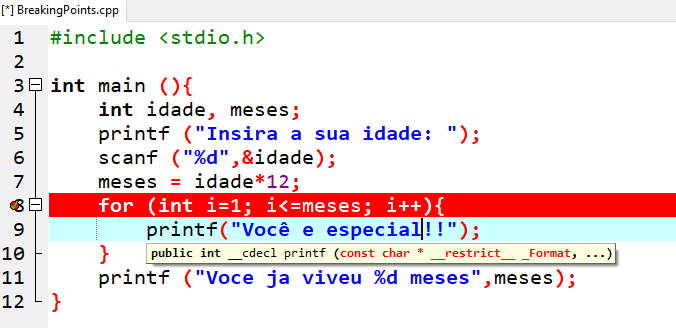
**Breaking Points**

Existem algumas formas de escrever códigos em algoritmos. Na verdade, cada programador tem sua particularidade e lógica, por isso dos tópicos de boas práticas e práticas padrão para programação, para que “qualquer” dev (*development =* desenvolvedor) consiga entender e fazer manutenção no seu código.

Imagine que por algum motivo o compilador não leia um **for**, um **if,** um **do while**, ou qualquer outra estrutura do seu algoritmo, por qualquer motivo desconhecido que seja. Se você estiver escrevendo o seu código em uma plataforma que aponte os erros, ela não identificará nesse caso, pois não é um erro de sintaxe, ou estrutural em si. Também temos a seguinte situação, a estrutura e a compilação estão ok, tudo roda normalmente, mas o resultado da execução não é o esperado. Dependendo da estrutura do código fica difícil identificar o que de fato está dando errado, se você apenas inverteu alguma lógica no algoritmo por exemplo ou se na correria você trocou algo na estrutura. Daí vem os breaking points! Eles são a sinalização de que o seu compilador precisa interromper a leitura e execução do código naquele ponto e apresentar o que foi encontrado até então, como variáveis e seus valores, mensagens de alertas e erros, ou seja, o breaking point auxilia no teste de funcionalidade do código. O breaking point irá ajudar a encontrar com maior facilidade e agilidade o problema que faz sua variável aparecer com valor diferente do esperado ou qualquer coisa do gênero, e assim você poderá corrigir o projeto e dar sequência ao trabalho.

Fica como na figura abaixo a representação do breaking point no DEV C++ (editor de código C++ com console de execução), existem algumas variações de interface para interface de IDE, mas no geral, aparece uma sinalização em cima do número da linha e um destaque na linha inteira.



Assim a possibilidade de achar um erro é mais rápida, otimizando seu tempo e auxiliando em códigos maiores por exemplo.