

Tutorial para a Aprendizagem da Utilização do *Debugger* do DevC++ para Depuração do Código

Introdução

Possivelmente, na 1.ª aula teórica desta Unidade Curricular, foi apresentado o diagrama mostrado na Figura 1, tendo sido descritas e debatidas cada uma das fases que permitem chegar do problema ao programa final. Uma das tarefas a realizar, depois de criado o programa, é a respetiva compilação. É algo que é feito de forma automática pelo compilador, apresentando os erros de semântica ou sintaxe (erros de escrita ou de construção).

Ciclo de desenvolvimento estruturado

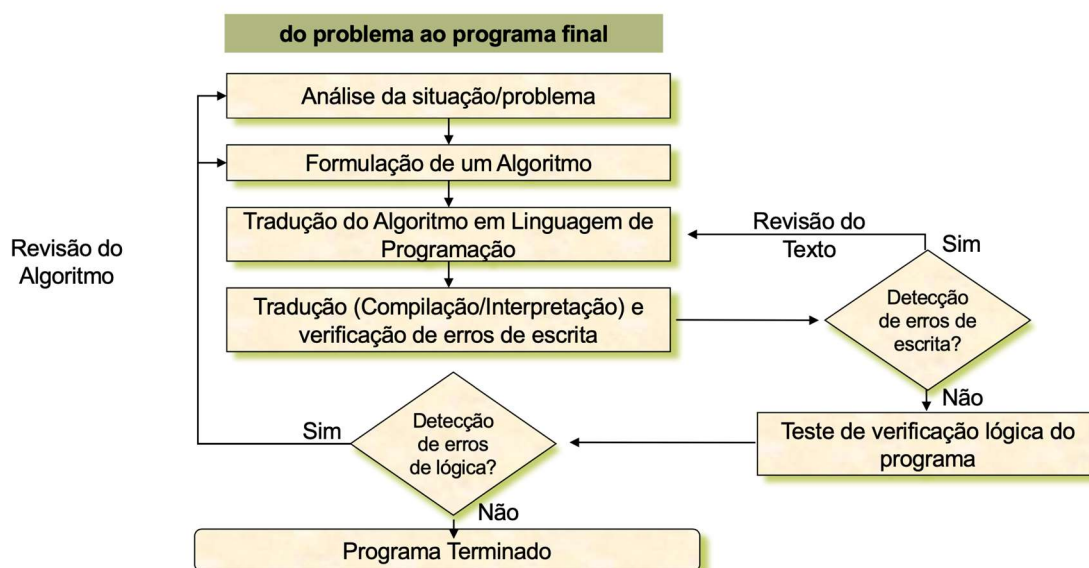


Figura 1 – Ciclo de desenvolvimento estruturado de um programa: do problema ao programa final (retirado de “Diapositivos da UC de AP”).

Cada Linguagem de Programação tem uma sintaxe e semântica própria, tal como acontece ao escrevermos em português, inglês ou outra língua: há regras de sintaxe e semântica a que devemos obedecer.

No caso de se estar a escrever um texto, usando um processador de texto (tipo WinWord), ele dispõe de um verificador de ortografia que assinala e propõe correções para palavras com erros de escrita (de semântica) e sugere até possíveis palavras para podermos corrigir o erro e, mais recentemente, já tem até alguma inteligência para detetar erros de sintaxe, tipo concordância de género e número e algumas outras falhas.

Como já devem ter percebido, o compilador também deteta falhas e propõe as correções que acha poderem dar uma ajuda.

Escola Superior de Tecnologia e Gestão de Viseu

UC: Algoritmos e Programação

Mas, e se o programa já não tem erros de compilação e, fazendo vários testes da sua execução na resolução do problema para que foi criado (a operação representada pela caixa “Teste de verificação lógica do programa” na figura), algum ou muitos dos resultados não estão conforme o que seria esperado e correto? Aí, como se vê no diagrama, o problema é bem mais grave, pois há que voltar ao início do processo ou ao 2.º passo (seta “Revisão do Algoritmo”). Está-se em presença de um ou mais erros de lógica, bem mais difíceis de detetar e corrigir e que implicam, eventualmente, reiniciar todo o trabalho de desenvolvimento da solução.

Aqui, o IDE (neste caso o DevC++) só pode dar uma ajuda, através de uma ferramenta denominada “*Debugger*” (depurador).

Como possivelmente já devem ter percebido, a única solução para se detetar estes erros é ir executando o código e, em certos locais ou linha a linha a linha, conseguir saber o valor de uma ou mais variáveis (para avaliar o valor nesse local) e também perceber quais as linhas que vão sendo executadas, isto no caso de haver estruturas de seleção (se...senão... ou se... então se....senão... ou caso...) ou de repetição (para ou enquanto) que possam ou não ser executadas.

O *debugger* permite especificar quais as variáveis cujo valor se vai visualizar e indicar o local de linhas de paragem e quando continuar a execução e de que forma. Os valores das variáveis vão sendo atualizados à medida que os respetivos valores vão sendo modificados, ao longo da execução do programa. Também são mostradas as linhas que vão sendo executadas, percebendo-se, assim, quais dos blocos de código são executados, sempre que existem estruturas de seleção ou repetição.

Claro que, se o *debugger* não for usado, podem-se sempre inserir linhas tipo “printf(“.....”), que permitam apresentar o valor de uma ou mais variáveis num dado local do programa ou perceber, com a apresentação de mensagens, se o programa passa por uma determinada linha ou entra numa determinada estrutura. Mas isto obriga a inserir o novo código e depois compilar e executar o programa até chegar à linha que mostra no ecrã o valor da variável que se pretende saber ou até chegar à zona que se pretende saber por onde “passa” a execução do programa. Ora, com o *debugger*, pode(m)-se especificar qual(is) a(s) variável(eis) de que se pretende(m) acompanhar o valor, à medida que o programa é executado, dispensando estar sempre a alterar o programa, recompilá-lo e reiniciar a sua execução. Até porque, se em qualquer momento for necessário apresentar o valor de uma nova variável, bastará indicar ao *debugger* qual ela é que, na execução da próxima linha, ela será acrescentada à lista dos valores mostrados.

Na outra vertente da depuração (o conhecer o fluxo de execução do programa), é possível colocar um “*breakpoint*” (ponto de interrupção: linha onde a execução do programa é suspensa) e depois executar o programa linha a linha, para saber por onde segue o fluxo de execução.

Estas funcionalidades facilitam bastante a tarefa de depurar o programa (na sua componente lógica, “o algoritmo”), pelo que é importante que se perceba como usá-las.

Depois deste preâmbulo, vai-se então dar início ao estudo do *Debugger* do DevC++ e a sua utilização prática.

Configuração do DevC++ para utilização do Debugger

Antes propriamente de iniciar a utilização do *Debugger* é necessário instruir o IDE DevC++ de que se pretende utilizar essa funcionalidade. É isso que vai ser o objetivo das tarefas a realizar na próxima secção.

Tarefas a executar:

1. Ir ao menu Ferramentas -> Opções do Compilador, conforme se mostra na Figura 2.

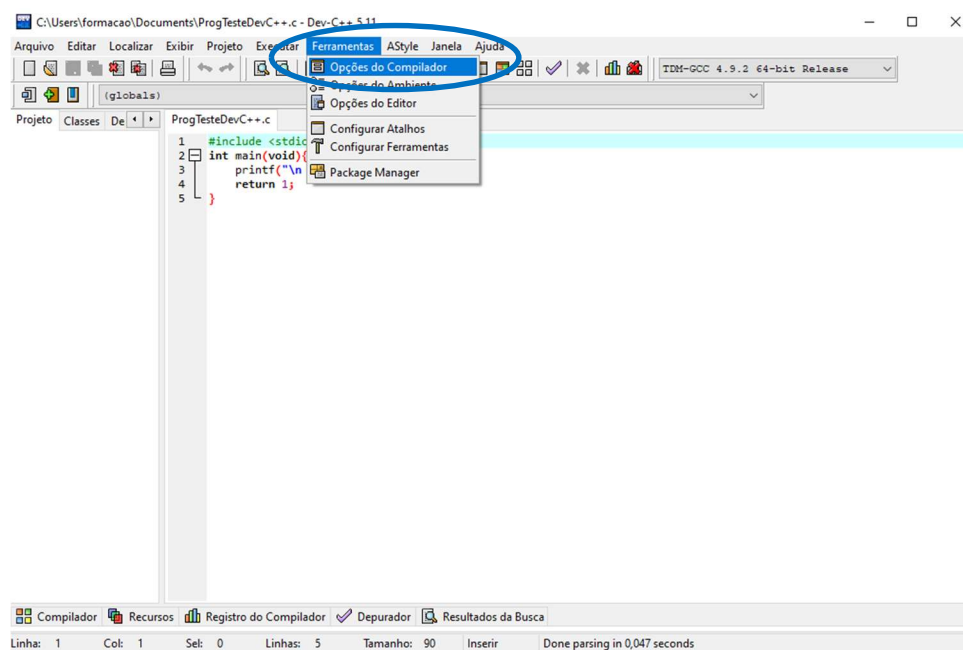


Figura 2 – Menu Ferramentas, para seleccionar “Opções do Compilador”.

2. Ao clicar em “Opções do Compilador”, abre-se uma janela, conforme se mostra na Figura 3 (está seleccionado o separador “Compilador”). Nesta, existe uma caixa de verificação (*Check Box*) de nome “Adicionar os seguintes comandos quando chamar o compilador” que deve ser seleccionada (ver a figura).

Na caixa imediatamente abaixo, inserir a seguinte diretiva de compilação: “-O0” (sinal de subtração, letra “O” maiúscula, zero), conforme está assinalado com a elipse.

Escola Superior de Tecnologia e Gestão de Viseu
UC: Algoritmos e Programação

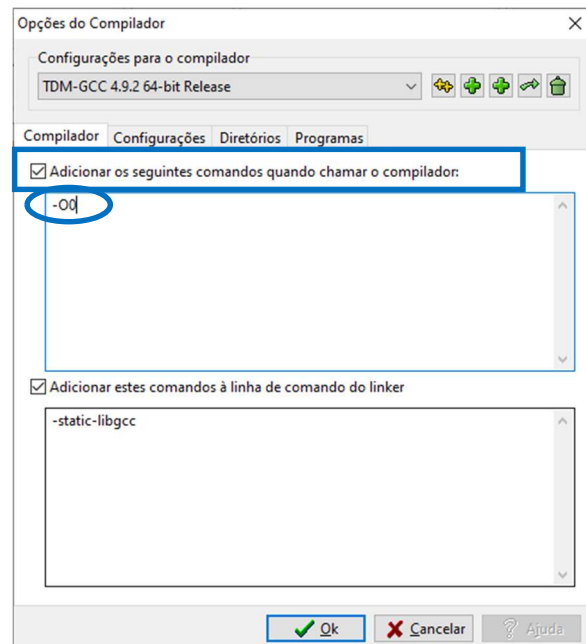


Figura 3 - Ecrã correspondente à funcionalidade "Opções do Compilador".

- Seguidamente, seleccionar o separador "Configurações", ficando a janela relativa às "Opções do Compilador" conforme se mostra na Figura 4. Seleccionar o separador "Linker" e deve certificar-se que a opção "Gerar Informação de depuração" esteja parametrizada como "Yes", conforme está assinalado com a elipse na figura. Caso assim não seja, clicar no "no" que estará lá e, depois, na ComboBox, colocar em "Yes".

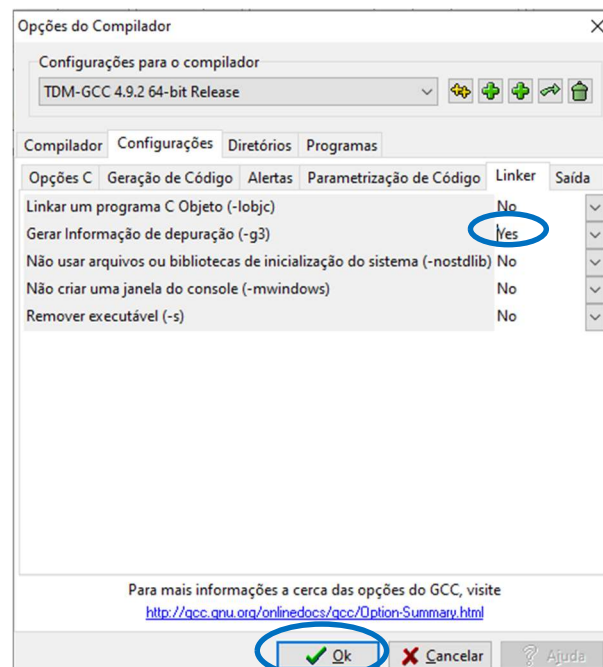


Figura 4 – Separador "Configurações" da funcionalidade "Opções do Compilador".

- Depois de realizadas estas operações, clicar no botão OK. A configuração para utilizar o Debugger está pronta. Pode-se agora iniciar a aprendizagem da sua utilização.

Como utilizar o Debugger para depurar o código no DevC++

Parte I - Preparar o ambiente para iniciar o processo de depuração

1. Criem uma pasta com um nome sugestivo à v/ escolha, atendendo a que o projeto que se irá criar se destina a perceberem como utilizar a ferramenta Debugger. Já perceberam ser uma ferramenta muito importante para procurar detetar (e depois solucionar) os problemas de erros de lógica, quer motivados pela não compreensão correta do problema, quer pelo facto da solução que encontraram (o respetivo algoritmo) não estar correta.
2. Iniciem a execução do DevC++ e criem um Projeto (Figura 5).

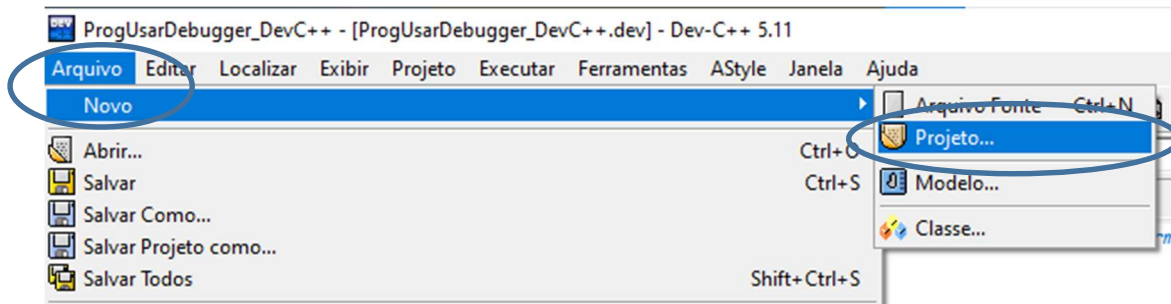


Figura 5 – Seleção da opção que permite criar um novo projeto no DevC++.

A sua execução resulta na criação do Projeto, conforme se mostra na Figura 6.

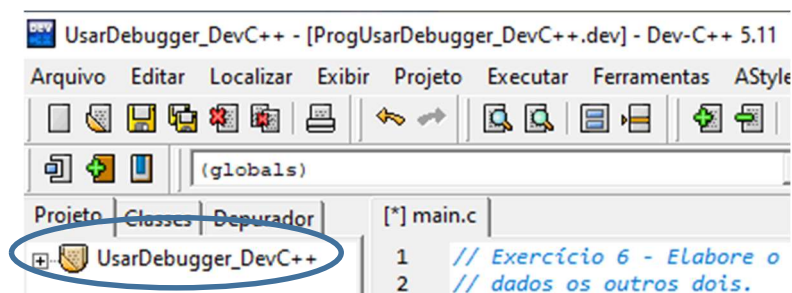


Figura 6 – Mostra do projecto criado e onde se irá inserir o programa que será usado para usar o debugger.

3. Criem agora um novo arquivo fonte no projeto existente, renomeando-o para “main”, pois vai ser o main (o programa que se irá usar para usar o *debugger*). Basta ir ao separador arquivo -> novo -> arquivo fonte. O resultado será algo como se apresenta na Figura 7.

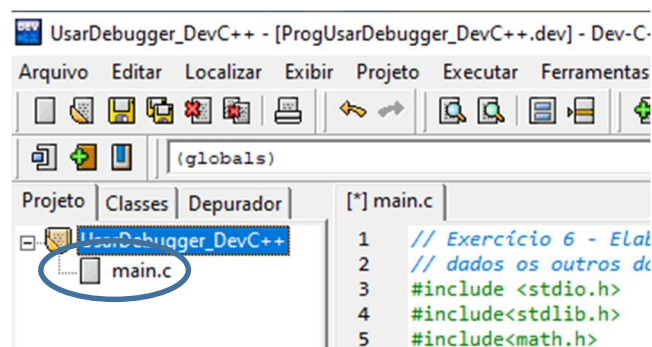


Figura 7 – Criação do arquivo main, um nome sugestivo, atendendo ao seu futuro conteúdo.

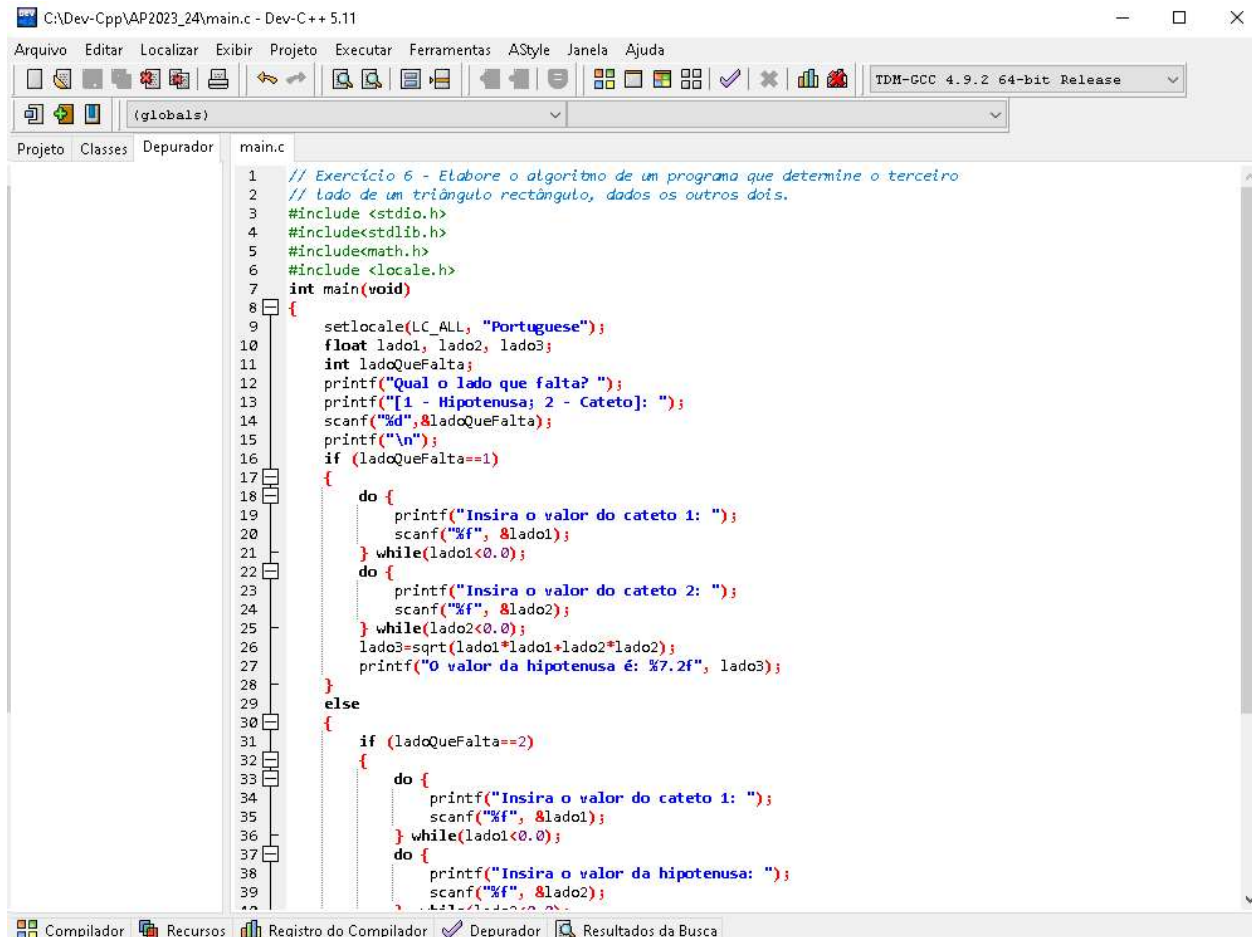
4. Copiem para o main, o código apresentado seguidamente:

// Exercício 6 - Elabore o algoritmo de um programa que determine o terceiro
// lado de um triângulo rectângulo, dados os outros dois.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <locale.h>
int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    float lado1, lado2, lado3; lado1=lado2=lado3=0;
    int ladoQueFalta;
    printf("Qual o lado que falta? ");
    printf("[1 - Hipotenusa; 2 - Cateto]: ");
    scanf("%d",&ladoQueFalta);
    printf("\n");
    if (ladoQueFalta==1)
    {
        do {
            printf("Insira o valor do cateto 1: ");
            scanf("%f", &lado1);
        } while(lado1<0.0);
        do {
            printf("Insira o valor do cateto 2: ");
            scanf("%f", &lado2);
        } while(lado2<0.0);
        lado3=sqrt(lado1*lado1+lado2*lado2);
        printf("O valor da hipotenusa é: %.2f", lado3));
    }
    else
    {
        if (ladoQueFalta==2)
        {
            do {
                printf("Insira o valor do cateto 1: ");
                scanf("%f", &lado1);
            } while(lado1<0.0);
            do {
                printf("Insira o valor da hipotenusa: ");
                scanf("%f", &lado2);
            } while(lado2<0.0);
            if (lado2*lado2-lado1*lado1 > 0){
                lado3=sqrt(lado2*lado2-lado1*lado1);
                printf("O valor do cateto: %.2f", lado3)}
            else
                printf("Triângulo Impossível!!!");
        }
        else
            printf("Tipo de cálculo inválido!!!");
    }
    printf("\n\n");
    printf("Para fechar a janela, digite <Return> ");
    c=getchar();
    c=getchar();
    return 0;
}
```

Decerto, já devem ter percebido que é o programa que resultou da resolução do exercício 6 da ficha n.º 1, com algumas alterações (inclusão de ciclos while para obrigar a que os valores inseridos sejam positivos).

A janela do programa ficará como se apresenta na Figura 8.



```

1 // Exercício 6 - Elabore o algoritmo de um programa que determine o terceiro
2 // lado de um triângulo rectângulo, dados os outros dois.
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <math.h>
6 #include <locale.h>
7 int main(void)
8 {
9     setlocale(LC_ALL, "Portuguese");
10    float lado1, lado2, lado3;
11    int ladoQueFalta;
12    printf("Qual o lado que falta? ");
13    printf("[1 - Hipotenusa; 2 - Cateto]: ");
14    scanf("%d", &ladoQueFalta);
15    printf("\n");
16    if (ladoQueFalta==1)
17    {
18        do {
19            printf("Insira o valor do cateto 1: ");
20            scanf("%f", &lado1);
21        } while(lado1<0.0);
22        do {
23            printf("Insira o valor do cateto 2: ");
24            scanf("%f", &lado2);
25        } while(lado2<0.0);
26        lado3=sqrt(lado1*lado1+lado2*lado2);
27        printf("O valor da hipotenusa é: %.2f", lado3);
28    }
29    else
30    {
31        if (ladoQueFalta==2)
32        {
33            do {
34                printf("Insira o valor do cateto 1: ");
35                scanf("%f", &lado1);
36            } while(lado1<0.0);
37            do {
38                printf("Insira o valor da hipotenusa: ");
39                scanf("%f", &lado2);
40            } while(lado2<0.0);
41            lado3=sqrt(lado1*lado1-lado2*lado2);
42            printf("O valor do lado que falta é: %.2f", lado3);
43        }
44    }
45    return 0;
46 }

```

Figura 8 – Janela com o programa que vai ser utilizado para familiarização com a utilização do debugger.

Parte II - Como controlar o processo de execução do programa

1. Para iniciar o Depurador, basta clicar no separador “Depurador”, conforme é apresentado na Figura 9 e abrir o separador Depurador, ficando a janela conforme é mostrado na Figura 10.

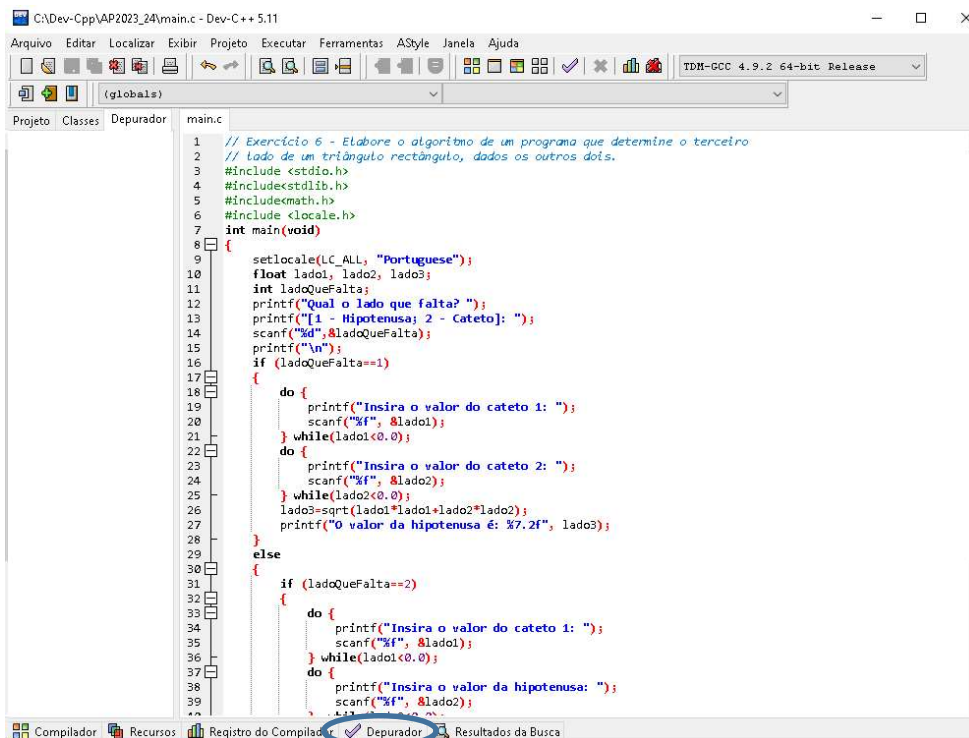


Figura 9 – Operação para iniciar o debugger (depurador).

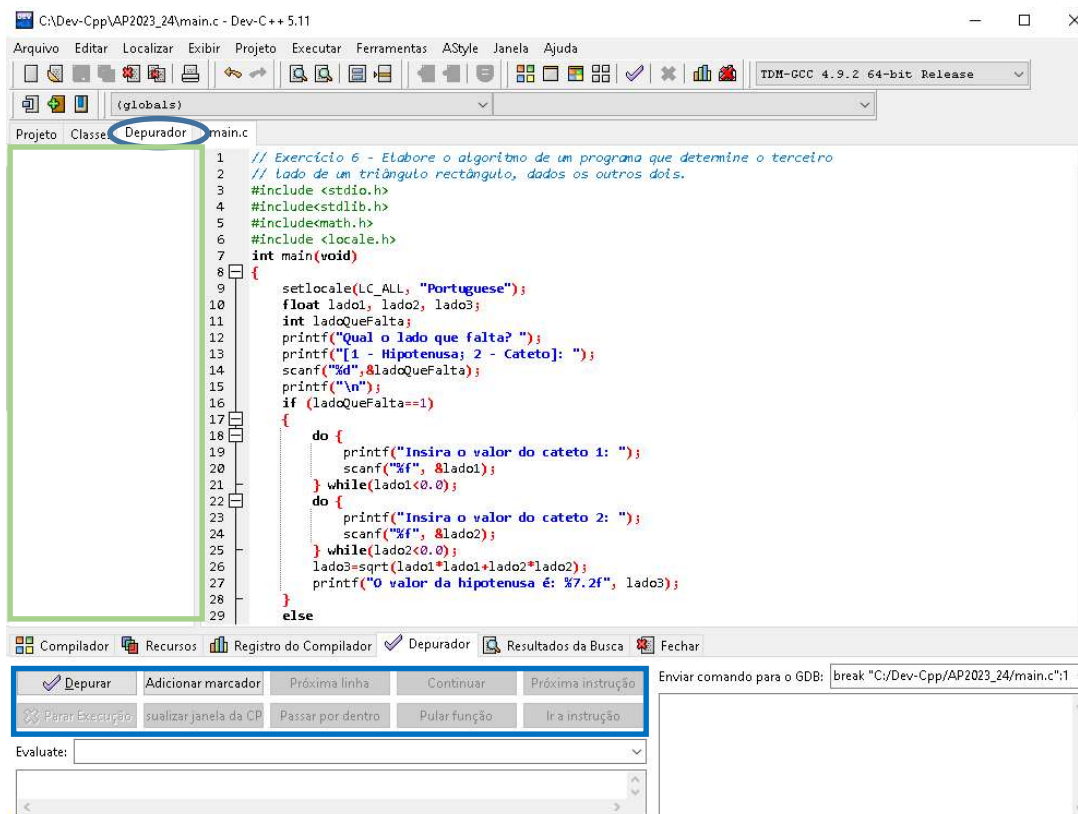


Figura 10 – Separador “Depurador”, onde aparecerão as variáveis cujos valores se pretendem conhecer, ao fazer-se o debug de um programa. Na parte inferior da janela, do lado esquerdo, as opções disponíveis quando se está a depurar o programa.

2. Na parte inferior da janela apresentada na Figura 10 (zona assinalada com o retângulo a azul), surgem agora as opções disponíveis para efetuar a depuração do programa, embora a maioria delas estejam, nesta fase, inativas.

De qq. modo, é importante, desde já, apresentar a lista das opções que serão descritas neste tutorial, pois são as de utilização mais comum:

- **Depurar:** Esta opção vai executar o programa em modo de depuração;
- **Para Execução:** Parará a execução do programa em modo de depuração;
- **Adicionar marcador:** Utilizada para adicionar uma nova variável de observação, ou seja, incluir no local do separador "Depurador" (na zona assinalada por um retângulo a verde) uma nova variável, cujo valor vai ser apresentado à medida que o programa for sendo executado, sempre com o seu valor actual;
- **Próxima Linha:** Executa a próxima linha do programa;
- **Continuar:** Continua a execução até ao próximo *breakpoint*.

Obs. : Como já foi referido atrás e se observa na figura, algumas destas opções só estarão ativas em modo de depuração.

3. Prosseguindo agora com a explicação de como depurar o código, é necessário inserir os *breakpoints* (pontos de paragem) nos locais onde se achar conveniente (aqueles onde se pretende verificar o "estado" das variáveis e a partir de onde se pretende saber quais as linhas que vão ser executadas a seguir).

Para inserir um *breakpoint*, basta clicar no número da linha onde se pretende inseri-lo. O processo mostra-se na Figura 11.

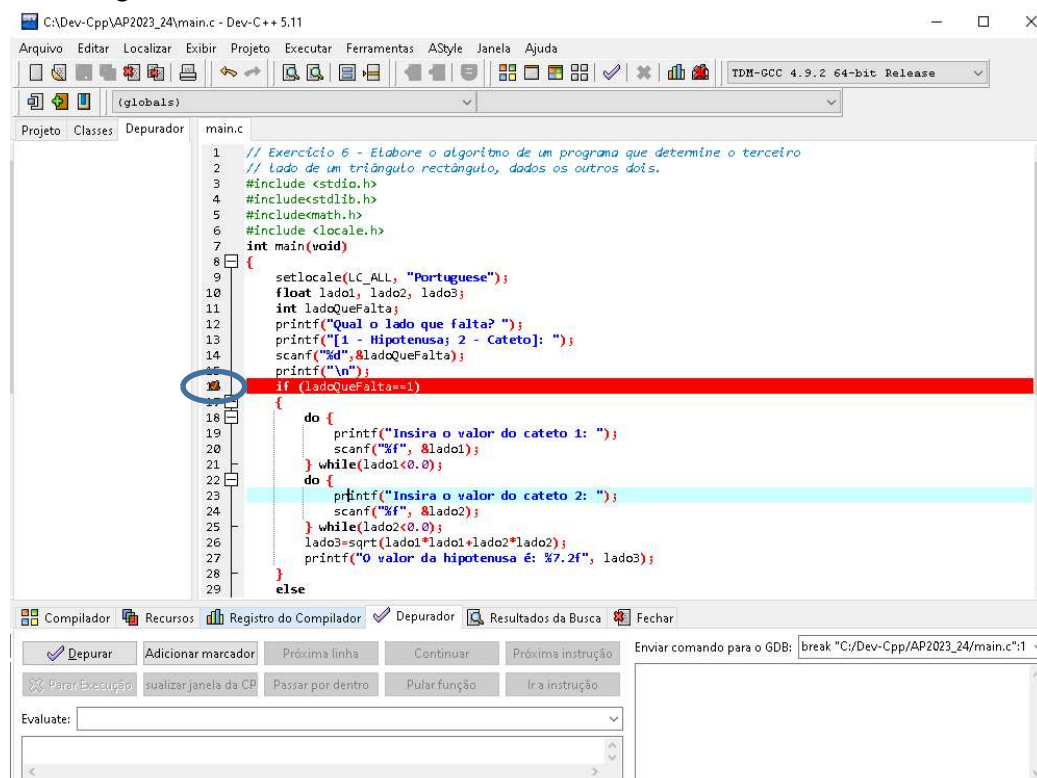


Figura 11 – Processo de inserção de um *breakpoint*, neste caso, na linha 16 e sinalizador respetivo.

4. Para executar o código em modo de depuração, basta clicar em Depurar, conforme se mostra na Figura 12. A Figura 13 mostra que o programa inicia a execução (abre uma nova janela – tipo ecrã de um monitor em modo consola – só capaz de mostrar caracteres, onde serão apresentadas as

Escola Superior de Tecnologia e Gestão de Viseu UC: Algoritmos e Programação

mensagens e aceites os dados introduzidos pelo utilizador, através do teclado), apresentando a mensagem “Qual o cateto que falta? [1 – Hipotenusa; 2 – Cateto]: “ e o cursor fica a sinalizar que está à espera que o utilizador responda com 1 ou 2, conforme o que este pretender calcular.

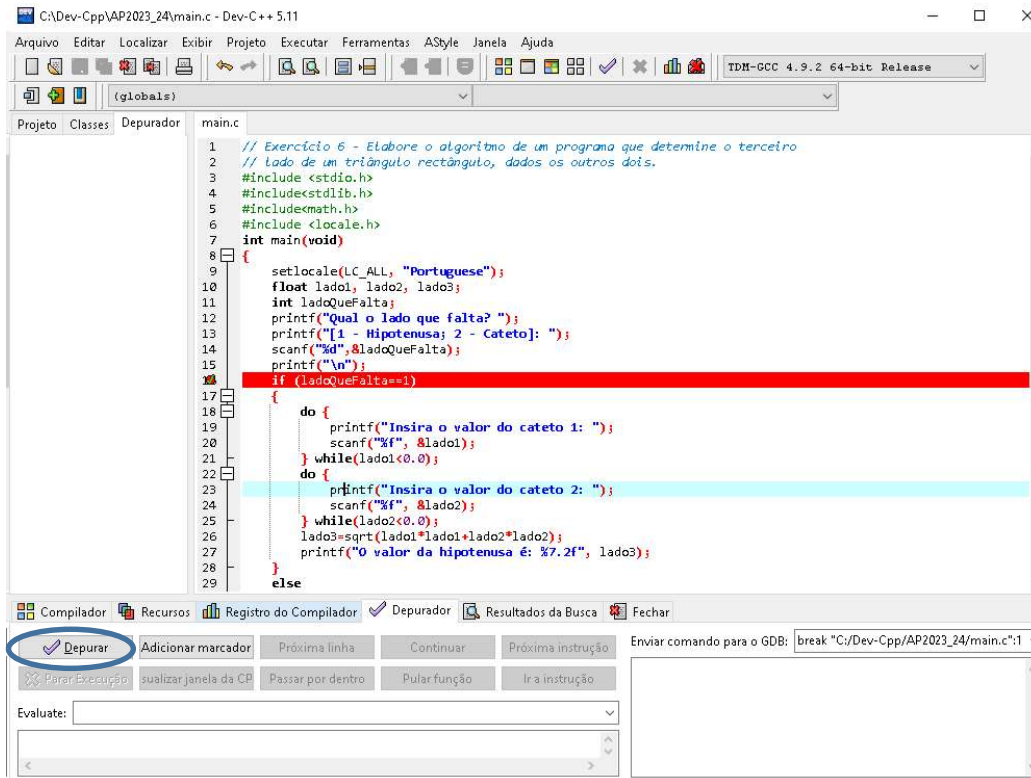


Figura 12 – Como iniciar o processo de debug (depuração) do programa.

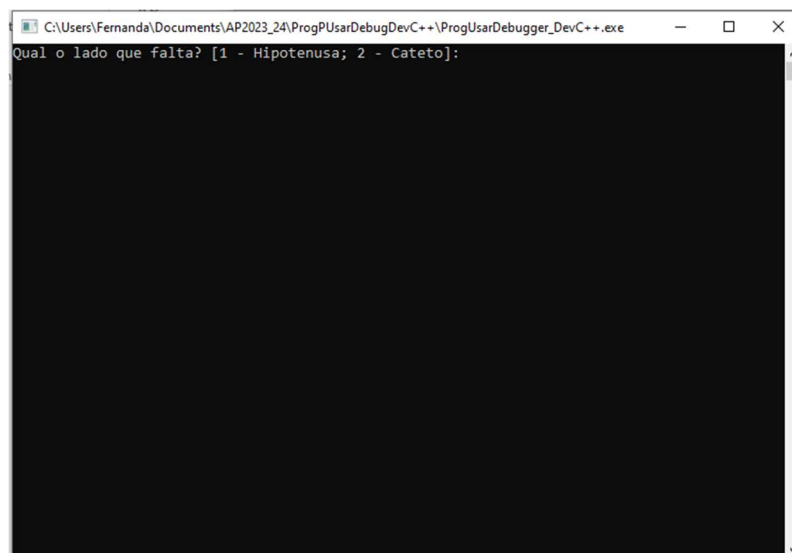


Figura 13 – Janela de execução do programa (tipo consola – em modo character).

5. Especificar 1 e digitar Enter (ver Figura 14). O programa aceita o 1, o cursor avança 2 linhas e a execução é suspensa (ver Figura 15). Porquê? Devido à existência do *Breakpoint*.

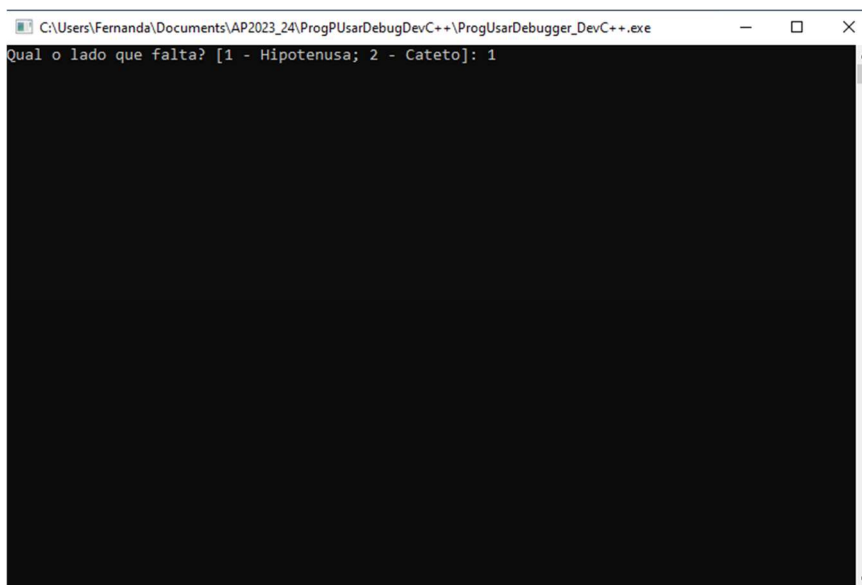


Figura 14 – O utilizador seleciona a opção 1, ou seja, pretende calcular a hipotenusa do triângulo retângulo, dados os 2 catetos.

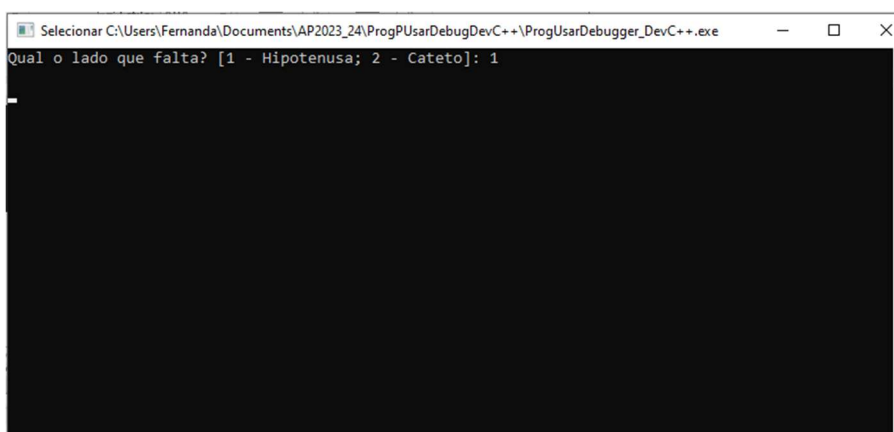


Figura 15 – A execução do programa é suspensa em resultado da existência do breakpoint.

Entretanto, na janela do programa, a paragem da execução na linha 17, é sinalizada pela linha a azul que está na linha do *breakpoint* (ver Figura 16).

Escola Superior de Tecnologia e Gestão de Viseu UC: Algoritmos e Programação

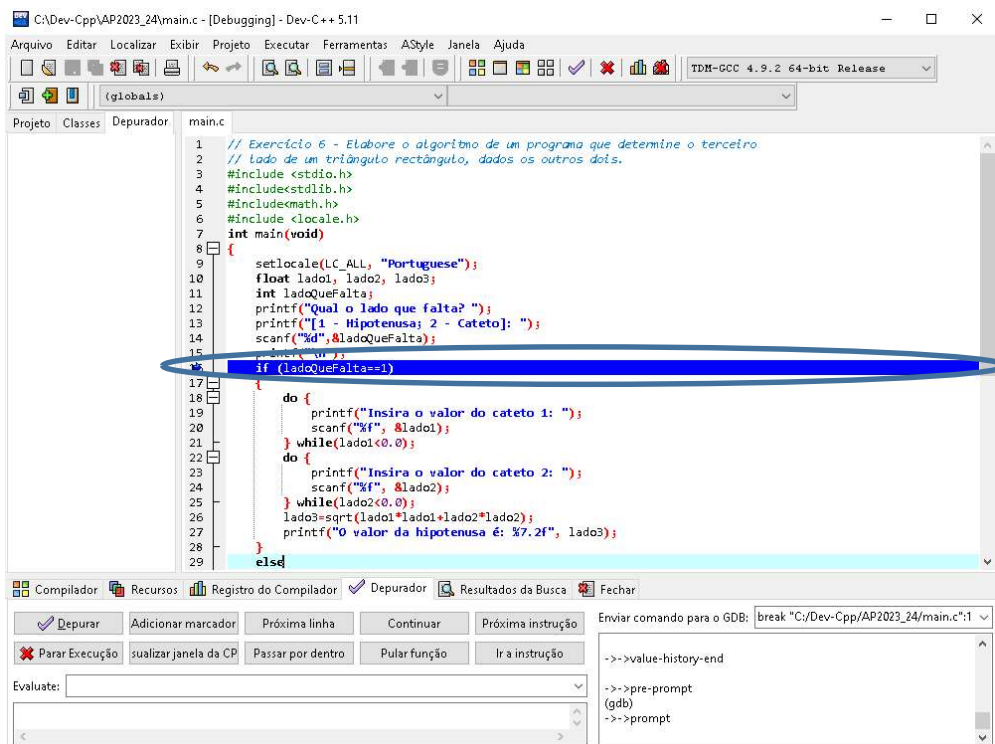


Figura 16 – Mostra da suspensão da execução do programa em resultado do breakpoint.

Seguidamente, deve-se alterar o tamanho da janela de execução e arrastá-la para outro local do ecrã, para que se possa clicar nos comandos do depurador. Depois, digitar em “Próxima linha” (ver Figura 17).

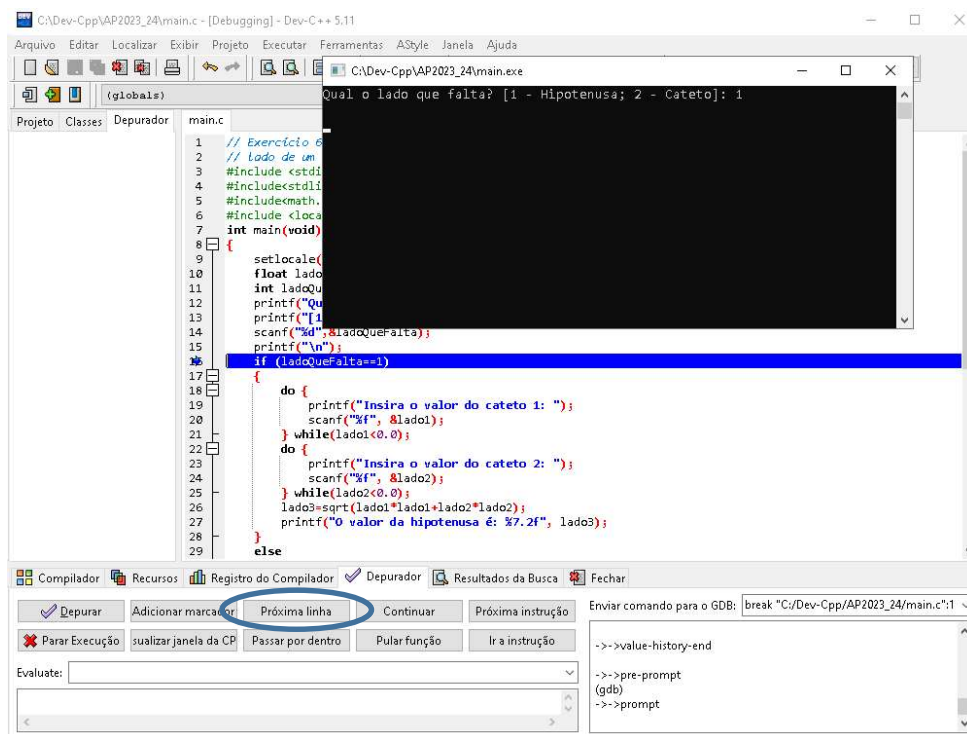


Figura 17 – Clicar no botão “Próxima Linha” para ser executada a linha seguinte do programa.

Como se pode observar, o programa entra na estrutura de seleção e, como a variável ladoQueFalta deve ter o valor 1, entrou no bloco controlado pela linha if (ladoQueFalta == 1), algo assinalado pela linha a azul que aparece na Figura 18.

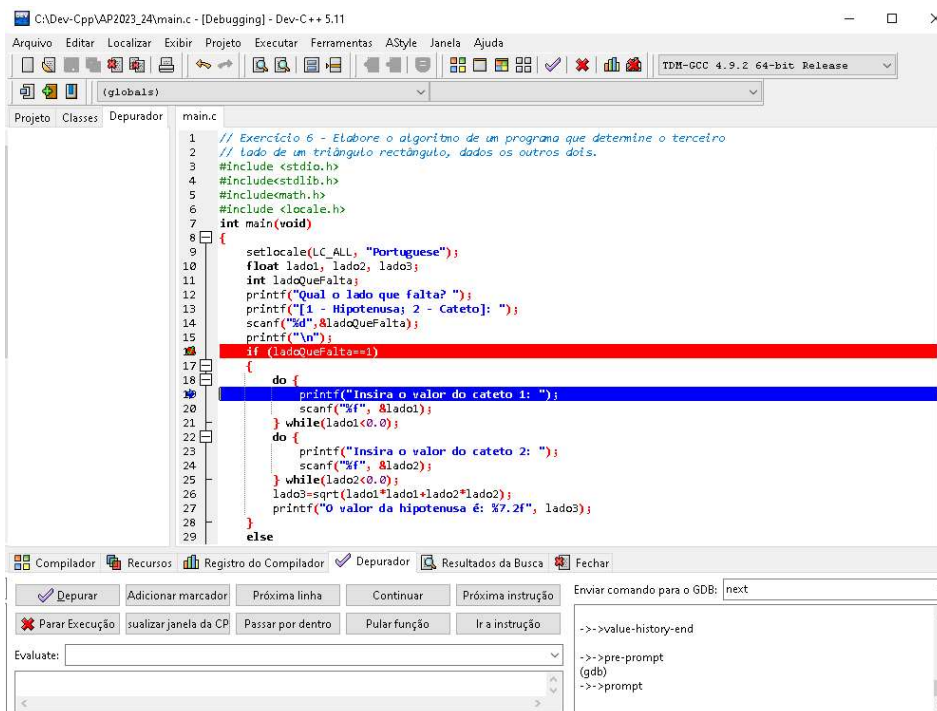


Figura 18 – A execução do programa entra no bloco do if, correspondente à condição ladoQueFalta==1.

Digitar novamente em “Próxima linha” e então a linha 20 é executada e surge no ecrã de execução a mensagem “Insira o valor do cateto 1: ” (ver Figura 19).

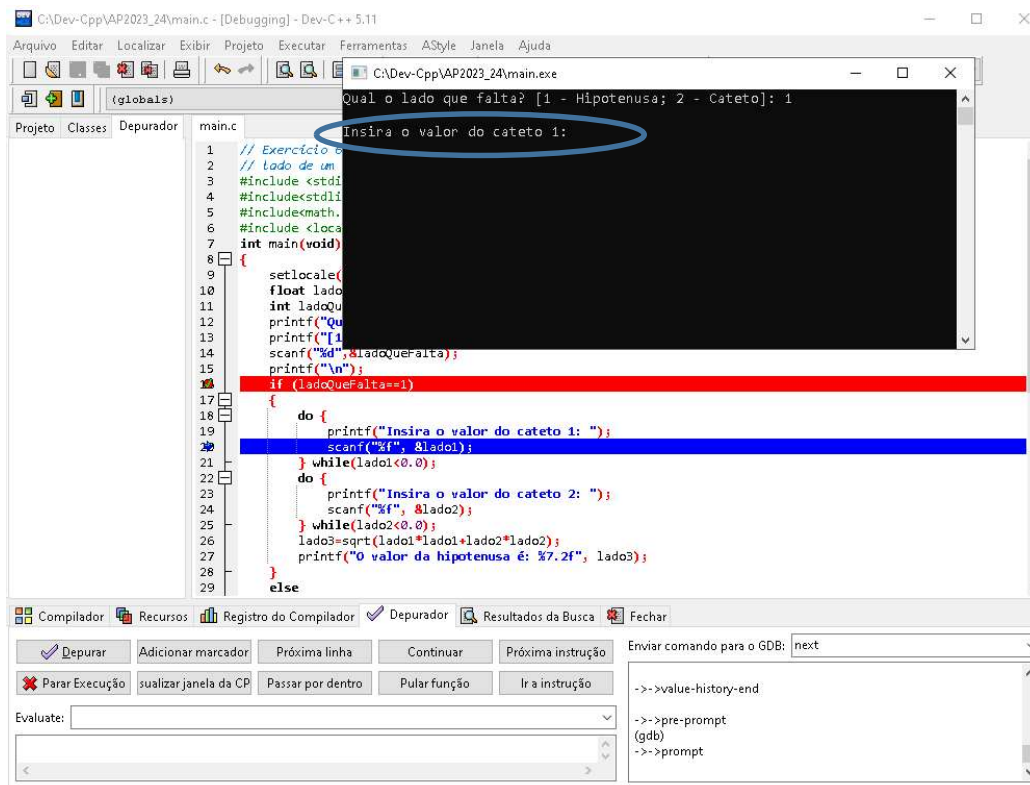


Figura 19 – É pedido o valor do 1.º cateto (está a ser executado o bloco do if, correspondente à condição ladoQueFalta==1).

Parte II - Como conhecer o “estado” do sistema (valor dos vários elementos de dados) quando o programa está em execução

Até agora, já se percebeu como se pode executar o programa passo a passo e ver o código que é realmente executado em resultado do estado do programa em qualquer linha. Mas, e como saber também o próprio “estado” em cada momento – o valor das variáveis e outros elementos de dados para efetuar a correção dos “bugs” (deste nome para os “erros” nasceu o nome “debug” e “debugger”)? O valor atual das várias variáveis em “observação” deve ser mostrado em qualquer linha em que o programa esteja suspenso em resultado da paragem por existir um *breakpoint*, ou porque se está a executar o programa linha a linha, p. ex.

1. Para explicar este segundo aspeto importante do processo de depuração, vai-se interromper a execução do programa em modo depuração, para iniciar novamente a sua depuração, desta vez, já com os valores das variáveis a serem apresentados. Para parar a execução, basta clicar no botão “Parar Execução” (ver Figura 20).

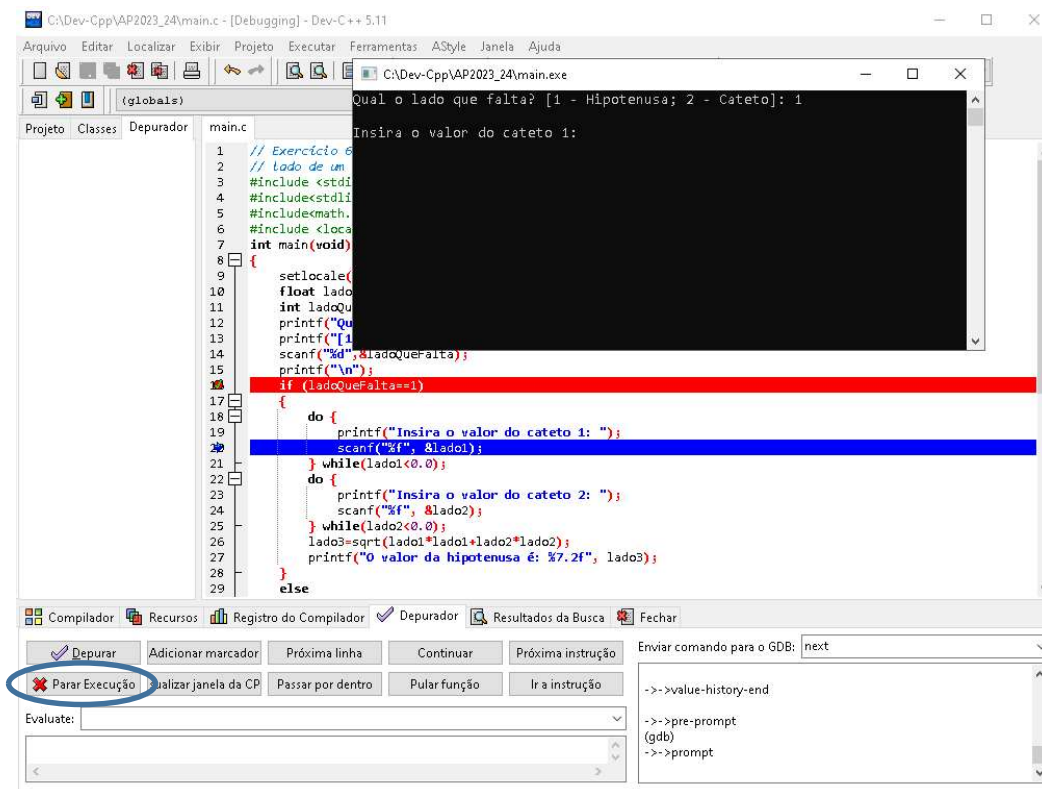


Figura 20 – Finalizar a execução do programa em modo depuração, clicando no botão “Parar Execução”.

2. Agora, suponha que pretendem saber os valores das variáveis `lado1` e `lado2`, ao longo da execução do programa. Para que tal aconteça, clicar no botão “Adicionar marcador”, conforme é mostrado na Figura 21. Surge uma janela de título “Nova Variável de observação” (sinalizada com o retângulo a azul), onde se pode inserir o nome da variável de que se pretende conhecer o valor a cada passo da execução do programa (neste caso, “`lado1`”).

Escola Superior de Tecnologia e Gestão de Viseu UC: Algoritmos e Programação

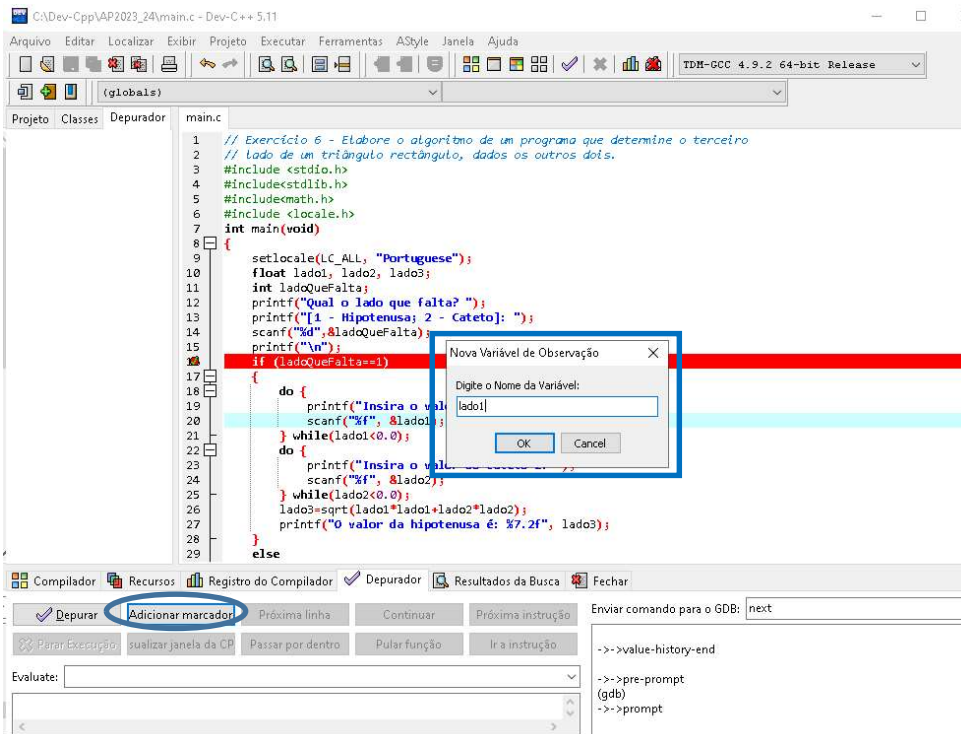


Figura 21 – Adicionar um marcador, ou seja, inserir uma nova variável cujo valor se pretende que seja mostrado aquando da execução do programa em modo de depuração. Neste caso, será apresentado o valor da variável “lado1”.

Como se pode observar na Figura 22, no separador “Depurador” (no lado esquerdo), surge agora uma linha “lado1 = Execute to evaluate” (sinalizada com o retângulo a azul), ou seja, quando o programa estiver a ser executado em modo “Depurar”, o valor da variável lado1 apresentado corresponde ao valor atual na linha onde a execução estiver suspensa. Assim, poderá saber-se o valor de uma variável em qualquer local do programa, sem obrigar a colocar linhas adicionais tipo “printf(“Valor da variável lado1 = %d\n”, lado1)”.

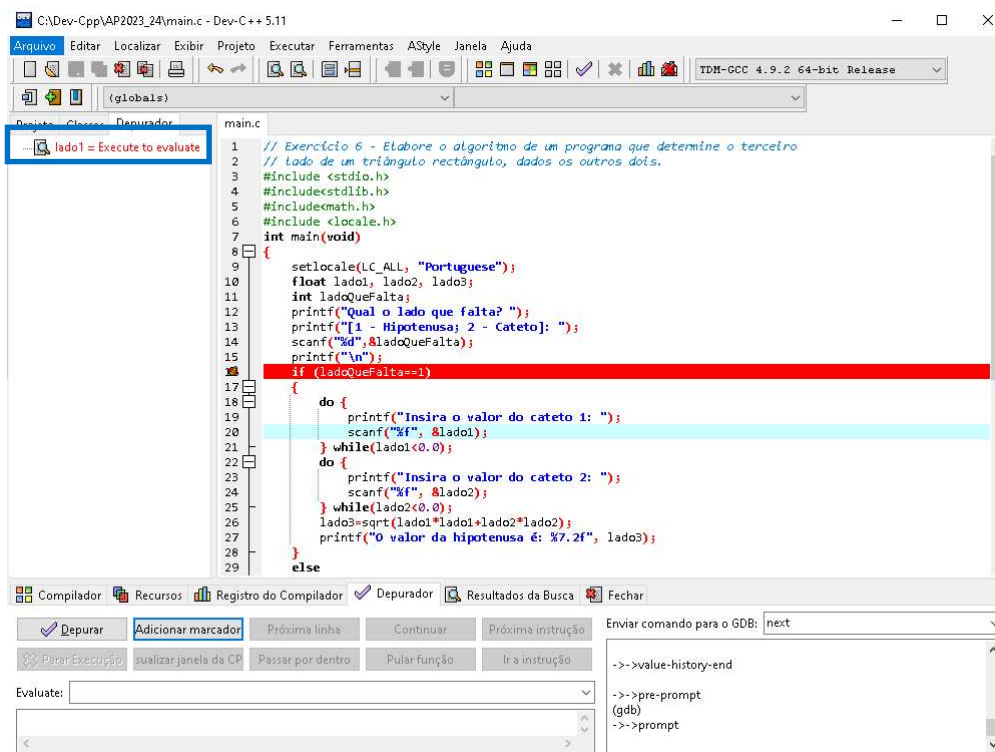


Figura 22 – Adição de um marcador para a variável lado1, sinalizada na zona do separador “Depurador” com a elipse azul.

3. Se for necessário saber o valor de mais do que uma variável, bastará repetir as operações referidas no item 9. Para conhecer o valor a variável lado2, clicar no botão “Adicionar marcador”, conforme é mostrado na Figura 23. Surge, de forma análoga, uma janela (sinalizada com o retângulo a azul), onde se pode inserir o nome da variável de que se pretende conhecer o valor a cada passo da execução do programa (neste caso, “lado2”), sendo, obviamente, apresentada no separador Depurador, conforme mostra a Figura 24.

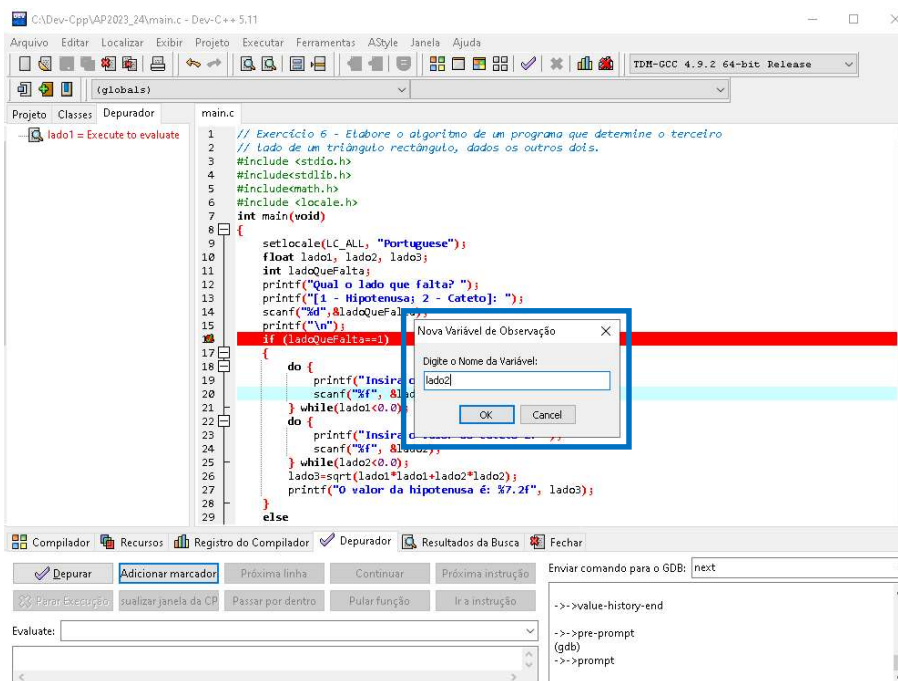


Figura 23 – Adicionar um marcador, ou seja, uma variável cujo valor se pretende que seja mostrado aquando da execução do programa em modo de depuração. Neste caso, será apresentado o valor da variável “lado2”.

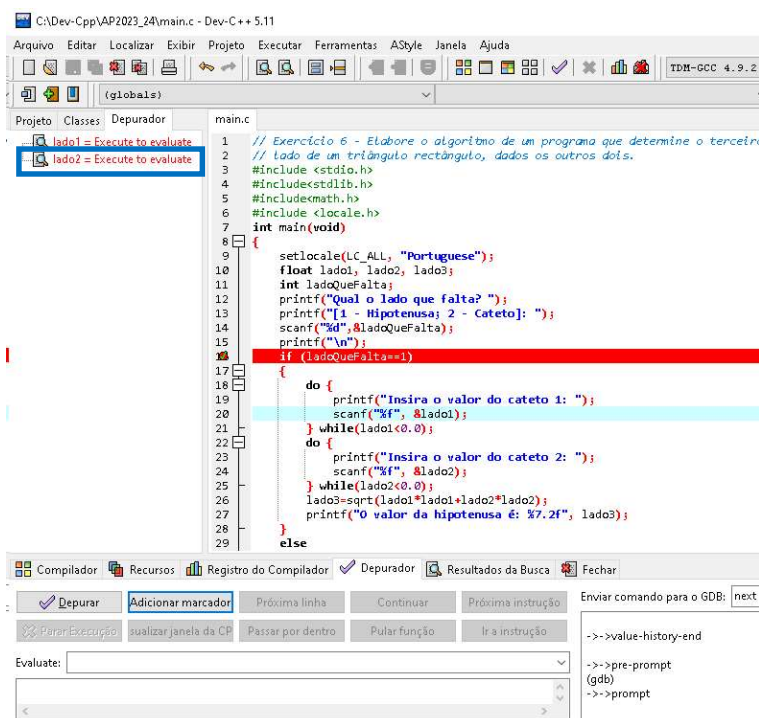


Figura 24 – Adição de um marcador para a variável lado2, sinalizada na zona do separador “Depurador” com o rectângulo azul.

4. Vai-se agora testar a nova funcionalidade implementada, ou seja, coloca-se em execução o programa em modo Depurar, clicando para isso no botão respetivo, assinalado com o retângulo a azul (ver Figura 25). Imediatamente, surge a janela de execução com a 1.^a mensagem do programa. Como se pode observar na Figura 26, depois de ser especificado 1, o programa é executado até ao *breakpoint* (suspendendo a execução) e, aí, o valor das variáveis em observação, lado 1 e lado2, é imediatamente apresentado (assinalado pelo retângulo a azul), neste caso, 0, Clicar em “Próxima linha” (assinalado com o retângulo a verde), sendo então executada a instrução `printf(“Insira o valor do cateto 1: “)` (ver Figura 27).

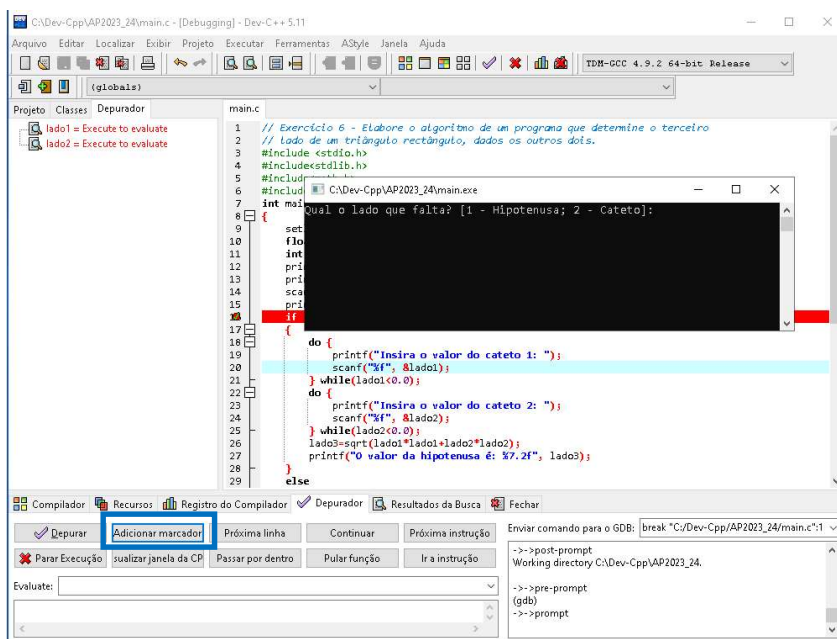


Figura 25 – Colocação em execução do programa em modo Depuração, clicando no botão “Depurar” (assinalado com o retângulo azul), surgindo a janela de execução do programa.

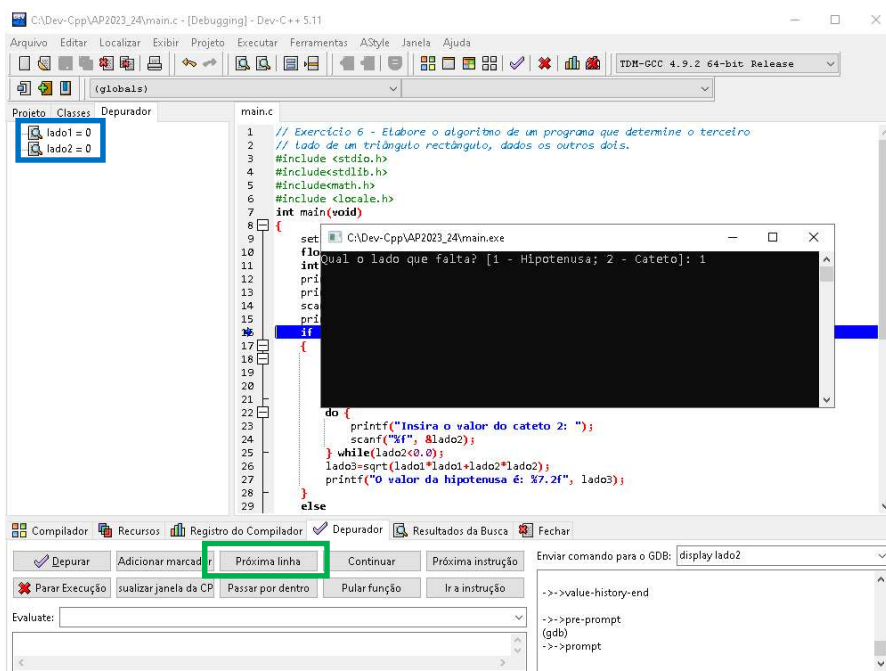


Figura 26 - Depois de ser iniciada a execução do programa em modo de depuração, e depois de ser especificado 1, o programa prossegue até ser suspenso no “breakpoint”. Então, na zona do depurador, os valores das variáveis lado1 e lado2 são apresentados (assinalados com o retângulo a azul).

Escola Superior de Tecnologia e Gestão de Viseu

UC: Algoritmos e Programação

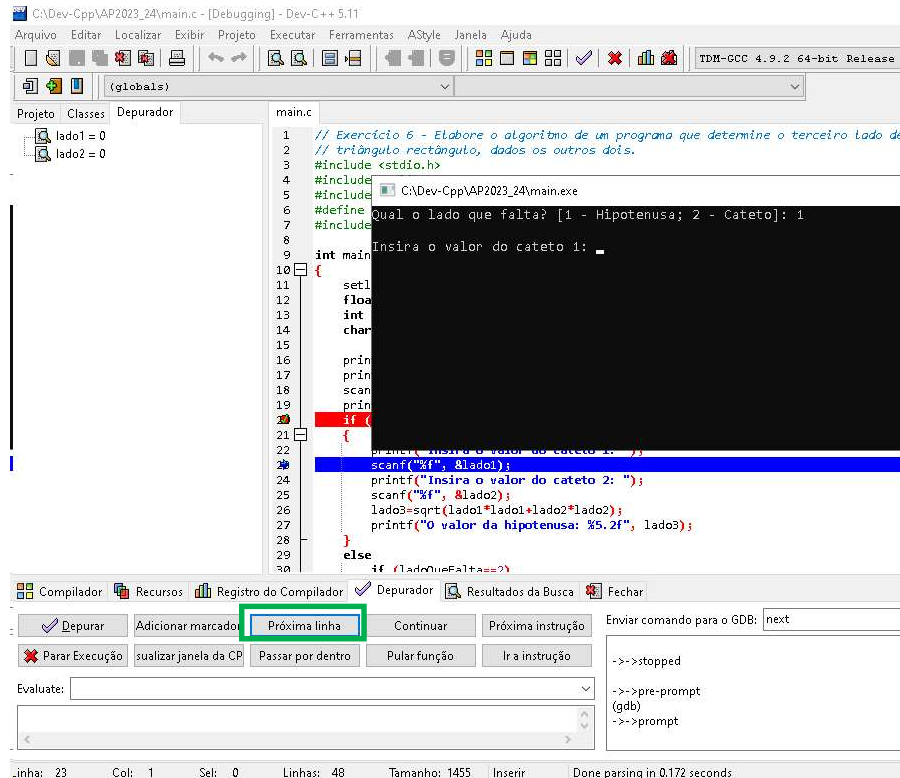


Figura 27 – A linha a azul assinala que na janela de execução se está à espera que seja inserido o valor do cateto 1.

- Na janela do DevC++, Clicar em “Próxima linha” e passa a ser aceite o valor do cateto 1 na janela de execução. Especificar um valor (neste caso, foi 4). Aí, o valor da variável “lado1” mostrada na zona do depurador, é atualizado, passando a apresentar o valor 4, conforme pode ser observado na Figura 28.

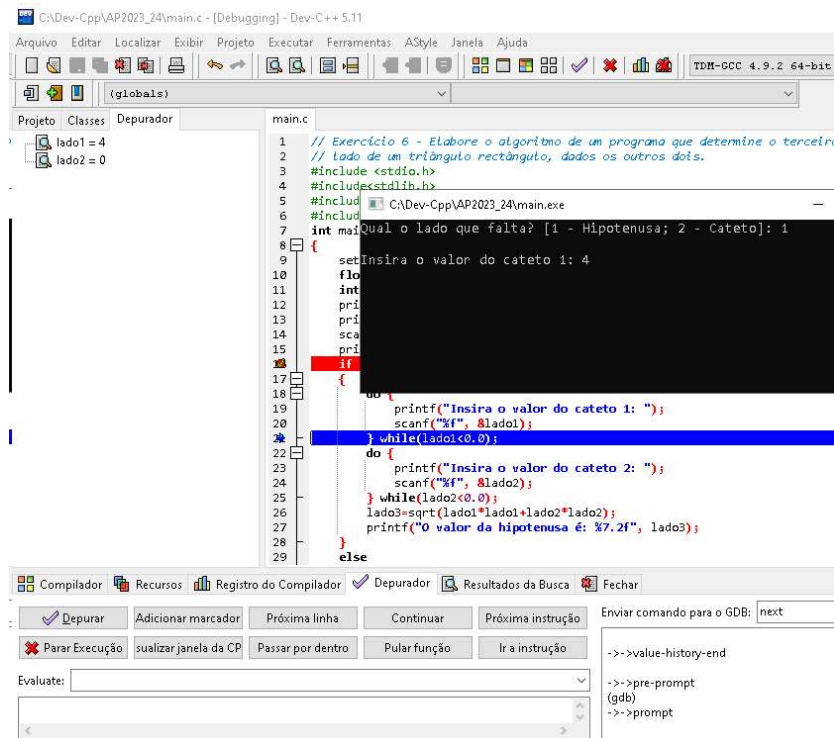


Figura 28 – Quando foi inserido 4 para o valor do cateto 1, o valor de lado 1, na zona do depurador, foi atualizado para 4, como seria de esperar.

Escola Superior de Tecnologia e Gestão de Viseu UC: Algoritmos e Programação

6. Proceder de forma que o programa prossiga a sua execução até ser aceite o valor do cateto 2 (ver Figura 29). Neste caso, especificou-se o valor 3. Então, o valor da variável lado2 mostrada na zona do depurador, é atualizado, passando a apresentar o valor 3, conforme pode ser observado na mesma figura.

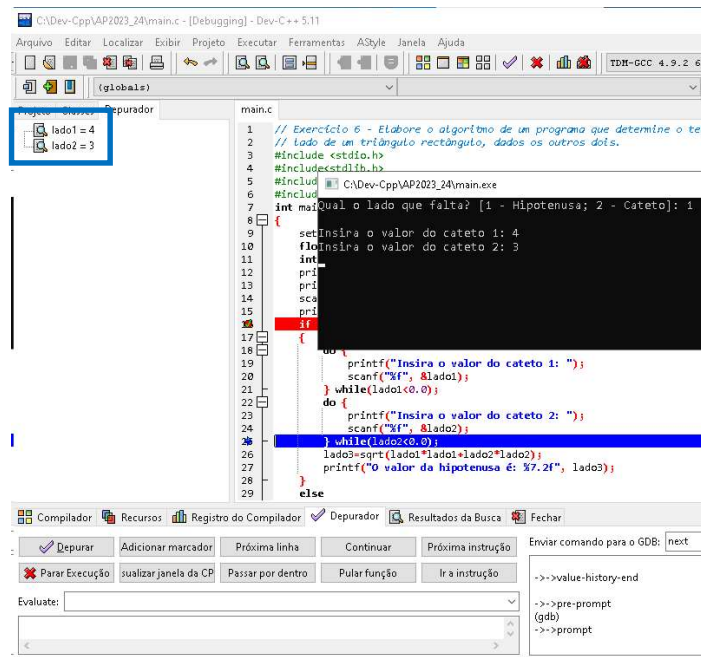


Figura 29 – Inserir o valor do cateto 2 (neste caso 3), sendo atualizado o valor de lado2, na zona do Depurador.

7. Vai supor-se que agora se pretende acrescentar uma nova variável para observação, neste caso, lado3. Bastará proceder da mesma forma usada para lado1 e lado2, pois pode acrescentar-se ou eliminar-se uma variável em observação em qualquer momento da execução do processo de depuração. Proceder conforme se pode observar na Figura 30. O resultado da operação apresenta-se na Figura 31, sendo mostrada a variável lado3 com o seu valor atual.

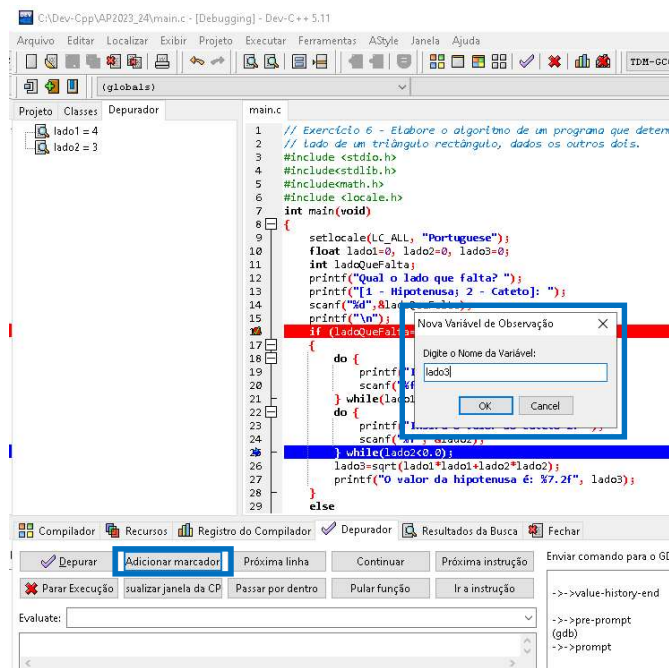


Figura 30 – Inserção de novo Marcador, neste caso, lado3 (que será, neste caso, o valor da hipotenusa), pois só agora se percebeu que vai ser necessário saber o seu valor.

Escola Superior de Tecnologia e Gestão de Viseu

UC: Algoritmos e Programação

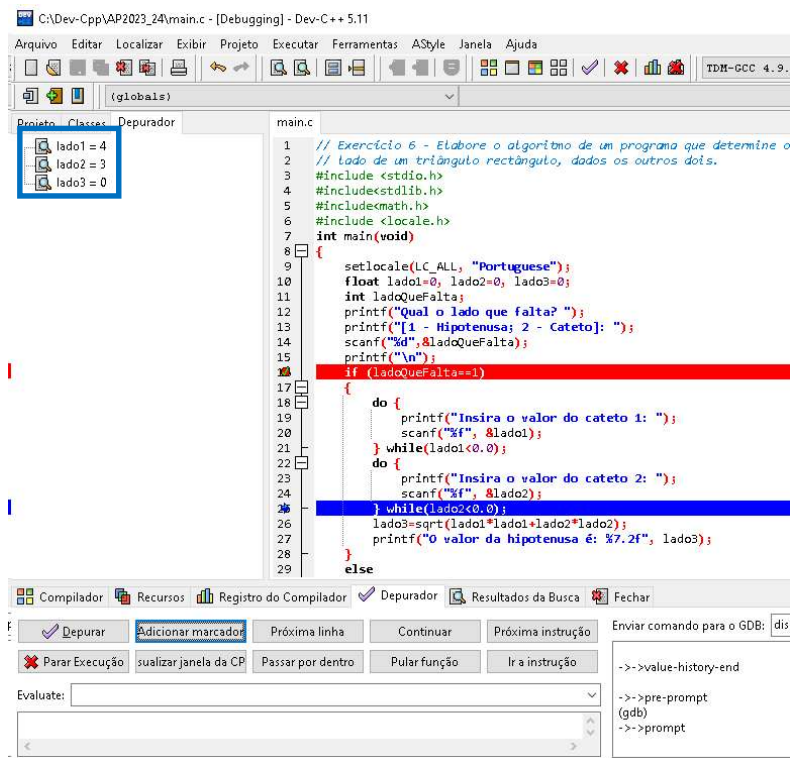


Figura 31 – Na janela do depurador aparece agora o valor do lado3 com valor atual (0).

8. Proceder de forma que o programa prossiga a sua execução até ser calculado e apresentado o valor da hipotenusa na janela do depurador, conforme se mostra na Figura 32 (basta avançar a execução até que a linha 26 seja executada). Depois de calculado o lado3, o seu valor já aparece na janela do depurador, conforme é mostrado na Figura 32.

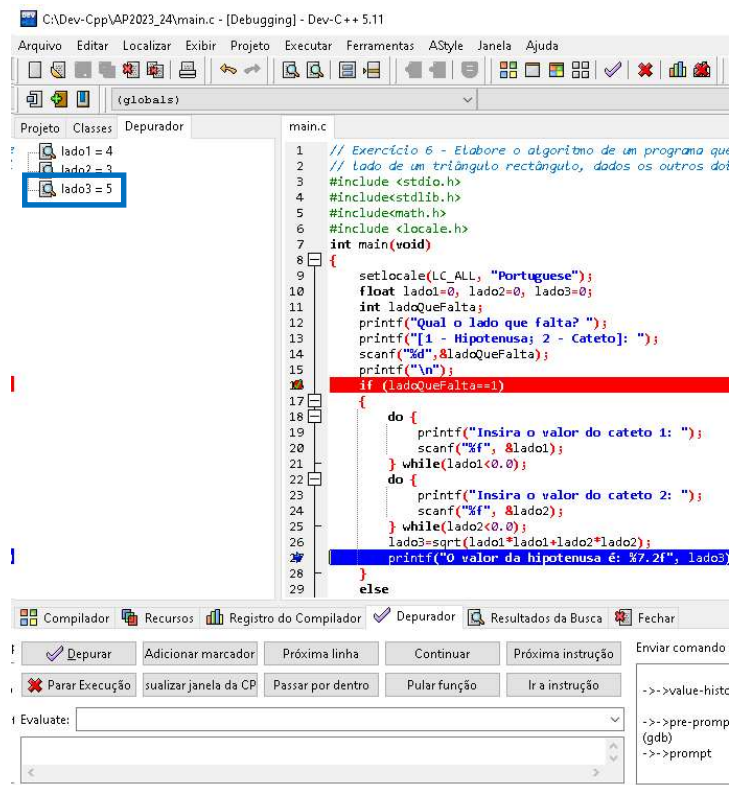


Figura 32 - Apresentação do valor do lado3 na zona do Depurador, mostrando, neste caso, o valor da hipotenusa. A inclusão da nova variável de observação, já durante a execução do programa em modo de depuração, é também possível.

Escola Superior de Tecnologia e Gestão de Viseu UC: Algoritmos e Programação

9. Proceder agora de forma que o programa prossiga a sua execução até ser apresentado o valor da hipotenusa no ecrã de execução, conforme se mostra na Figura 33.

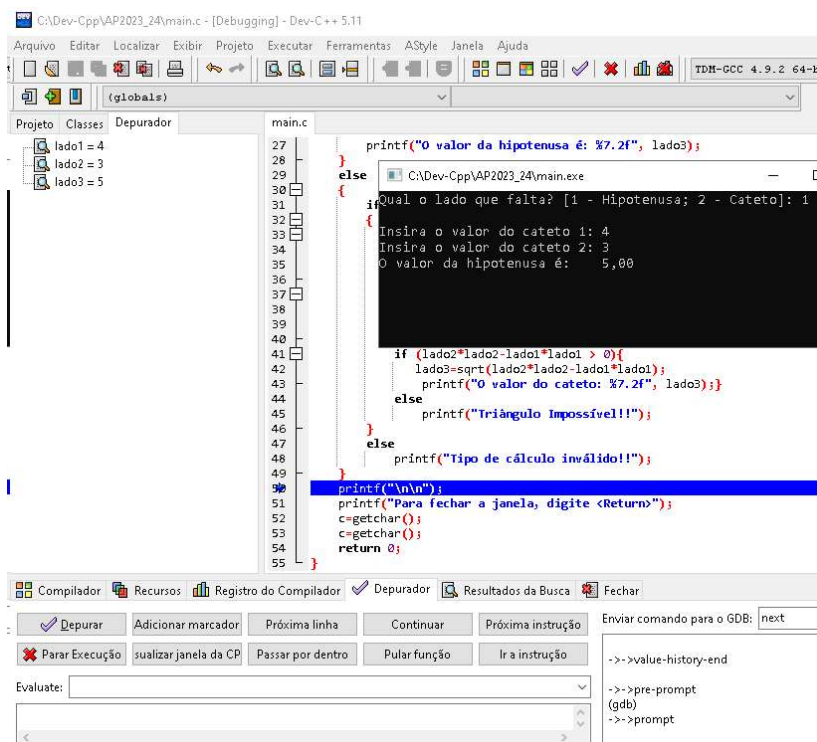


Figura 33 - Apresentação do valor da hipotenusa na janela de execução, em resultado da execução da linha 27 do programa.

10. Inserir um *breakpoint* na linha 52. De seguida, clicar em “Continuar” (ver Figura 34). O que acontece é que a execução continuará até à execução da linha onde está o novo *breakpoint*, sendo mostrada a mensagem a janela de execução, conforme se pode observar na Figura 35.

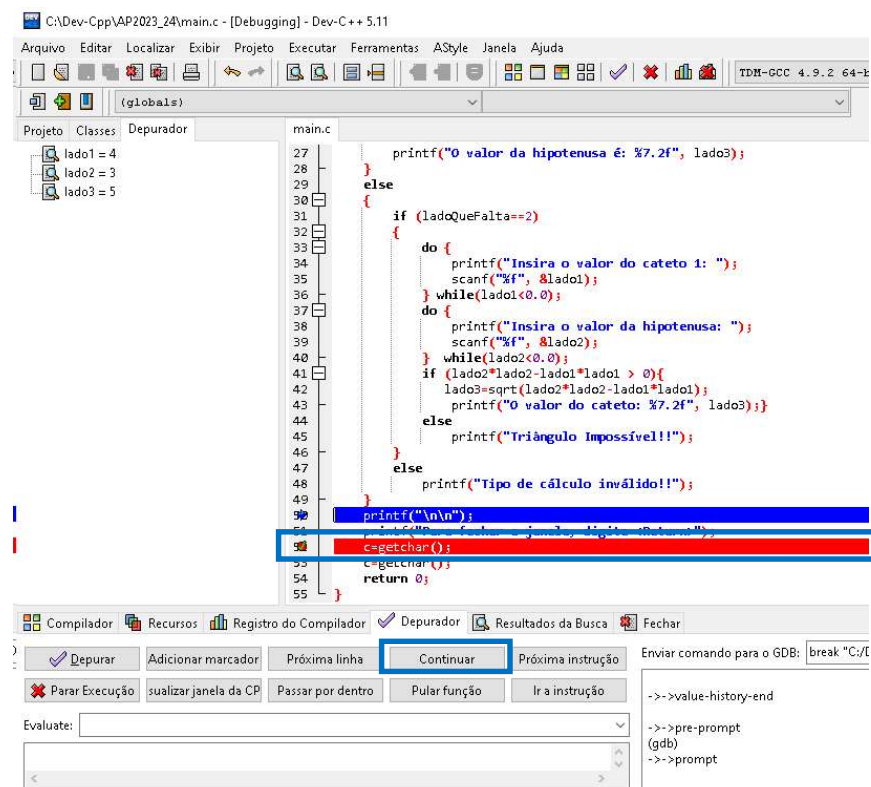


Figura 34 – Inserção de um novo *breakpoint* na linha 52 (assinalado pelo retângulo a azul).

Escola Superior de Tecnologia e Gestão de Viseu

UC: Algoritmos e Programação

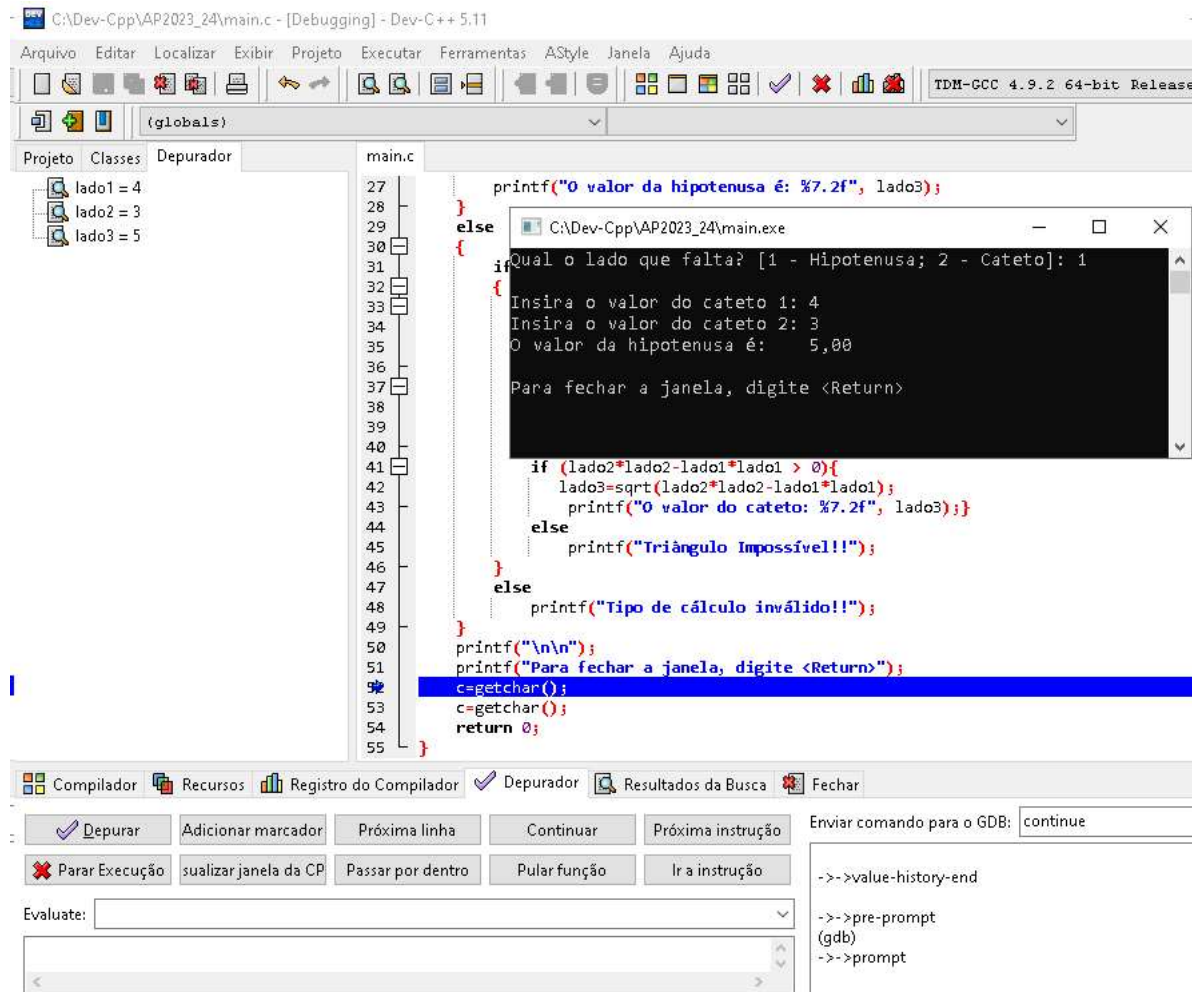


Figura 35 - Clicar no botão "Continuar". A execução continuará sendo mostrada a mensagem "Para fechar a janela, digite <Return>" e sendo suspensa a execução na linha 52.

Para finalizar este tutorial, importa referir que há alguns vídeos e outra informação que podem ser encontrados no Google e que abordam o tema da utilização do *debugger*, podendo, eventualmente, esclarecer alguns pormenores não tratados neste documento.