



FUNÇÕES E PROCEDIMENTOS

ECM404

FUNÇÕES

- ❑ Funções são blocos de código que podem ser nomeados e chamados em um programa:
 - ❑ **printf()** – função que escreve um valor na tela;
 - ❑ **scanf()** – função que lê um valor digitado no teclado.
- ❑ **Estruturação:** programas grandes e complexos são constituídos bloco a bloco;
- ❑ **Reutilização:** usar funções evita a cópia desnecessária de trechos de código que realizam a mesma tarefa.

FUNÇÕES

```
float media(int x, int y, float z);

int main (){
    int a,b;
    float m;
    printf("Digite o valor de a: ");
    scanf("%i",&a);
    printf("Digite o valor de b: ");
    scanf("%i",&b);
    m = media(a,b,2.0);
    return 0;
}

float media(int x, int y, float z){
    return (x+y)/z;
}
```

FUNÇÕES – PROTÓTIPO DA FUNÇÃO

❑ Protótipo da função:

- ❑ Funções devem ser declaradas antes de serem utilizadas. Assim, definir globalmente os protótipos das funções, permite chamá-las no programa principal (ou dentro de outras funções).
- ❑ Itens obrigatórios: tipo da função, nome da função e tipo das variáveis.

```
float media(int x, int y, float z);
```

```
int main (){  
    int a,b;  
    float m;  
    ...  
}
```

FUNÇÕES – PROTÓTIPO DA FUNÇÃO

❑ Protótipo da função:

❑ Itens obrigatórios: tipo da função, nome da função e tipo das variáveis.

tipo da função nome da função tipo das variável



```
float media(int x, int y, float z);
```

```
int main (){  
    int a,b;  
    float m;  
    ...  
}
```

FUNÇÕES – CHAMADA DA FUNÇÃO

- ❑ Chamada da função media na função main:

```
float media(int x, int y, float z);

int main (){
    int a,b;
    float m;
    printf("Digite o valor de a: ");
    scanf("%i",&a);
    printf("Digite o valor de b: ");
    scanf("%i",&b);
    m = media(a,b,2.0);
    return 0;
}

float media(int x, int y, float z){
    return (x+y)/z;
}
```

FUNÇÕES – DECLARAÇÃO DA FUNÇÃO

❑ Declaração da função:

❑ Deve-se colocar:

- ❑ tipo da função;
- ❑ nome da função;
- ❑ tipo das variáveis;
- ❑ nome das variáveis;
- ❑ corpo da função (comandos);
- ❑ return.

```
...  
return 0;  
}  
  
float media(int x, int y, float z){  
    return (x+y)/z;  
}
```

FUNÇÕES – CORPO DA FUNÇÃO

- ❑ Corpo da função:
 - ❑ Variáveis locais: são aquelas que só tem validade dentro da função na qual foram declaradas.

```
...  
return 0;  
}  
  
float media(int x, int y, float z){  
    float m;  
    m = (x+y)/z;  
    return m;  
}
```


FUNÇÕES – RETURN

- ❑ Uma função pode ou não retornar um valor:
 - ❑ Se ela retornar um valor, alguém deverá receber este valor na chamada;
 - ❑ Uma função que não retorna um valor é chamada de **procedimento** e seu tipo deve ser declarado como **void**.

```
void nome da função ( parâmetros ){  
    sequência de comandos;  
    return; // pode-se omitir  
}
```

FUNÇÕES – RETURN

- ❑ O valor de retorno tem que ser compatível com o tipo declarado para a função. Pode-se retornar qualquer valor válido em C:
 - ❑ Tipos pré-definidos: **int**, **float**, **char** e **double**;
 - ❑ Tipos definidos pelo usuário: **struct**.
- ❑ Uma função pode ter mais que um **return**:
 - ❑ Quando o comando **return** é executado, a função termina imediatamente;
 - ❑ Todos os comandos restantes são **ignorados**.

FUNÇÕES – BOAS PRÁTICAS

- ❑ De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função. Isso assegura que a função possa ser utilizada nas mais diversas aplicações, garantindo a sua generalidade.
- ❑ Evite variáveis globais! Elas podem ser alteradas por todas as funções do programa. Entretanto, quando houver uma variável local como mesmo nome que a global, a função dará preferência para a variável local.

DEPURAÇÃO DE CÓDIGO

Como depurar para iniciantes absolutos

Artigo • 12/01/2022 • 8 minutos para o fim da leitura • 2 colaboradores



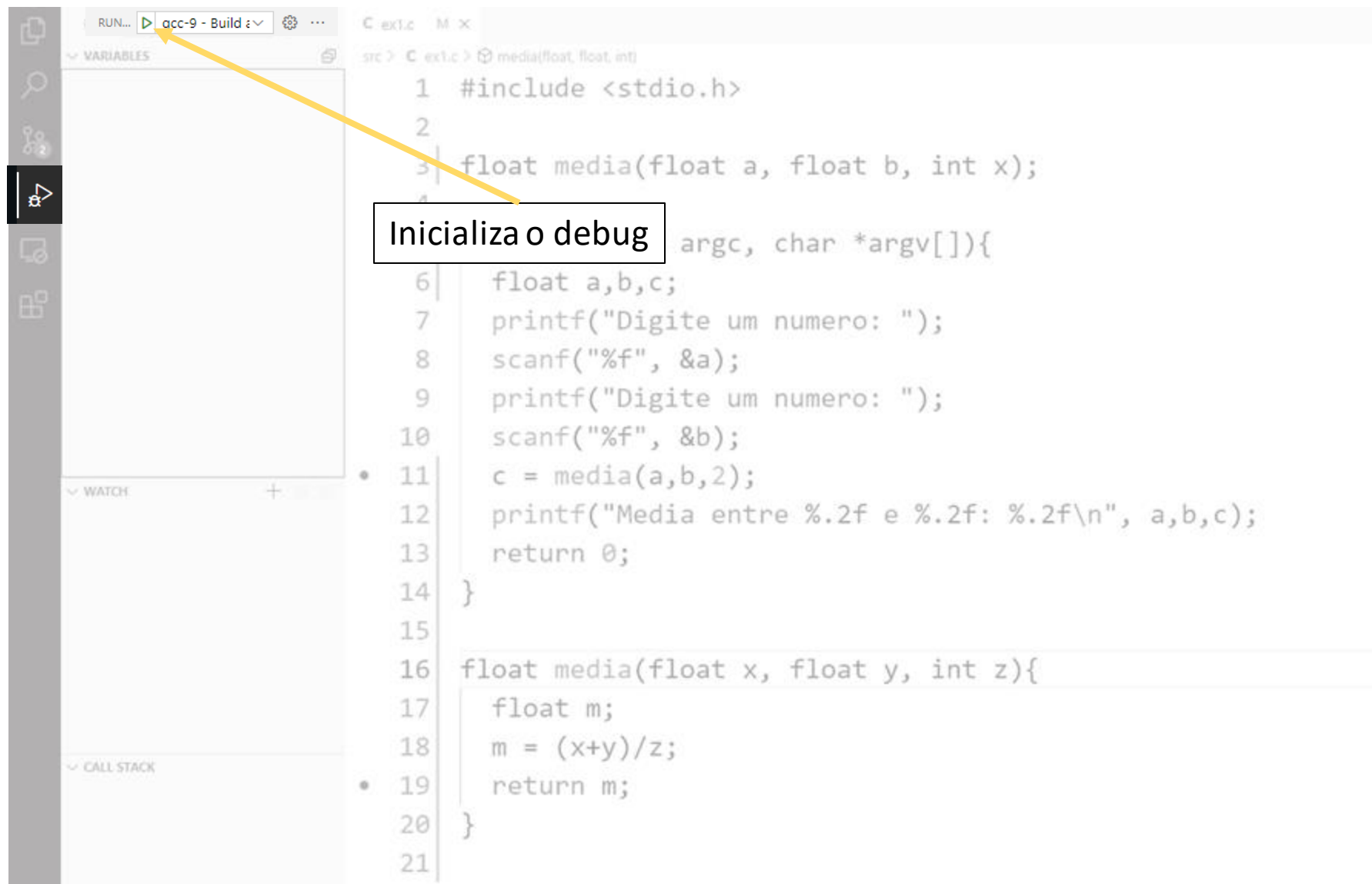
Sem falha, o código que escrevemos como desenvolvedores de software nem sempre faz o que esperamos. Às vezes, ele faz algo completamente diferente! Quando isso acontece, a próxima tarefa é descobrir por que e, embora possamos ficar tentados a simplesmente encarar nosso código por horas, é muito mais fácil e eficiente usar uma ferramenta de depuração ou depurador.

Um depurador, infelizmente, não é algo que possa revelar magicamente todos os problemas ou "bugs" em nosso código. *Depuração* significa executar o código passo a passo em uma ferramenta de depuração, como o Visual Studio, para localizar o ponto exato em que você cometeu um erro de programação. Você entenderá quais correções são necessárias para fazer em seu código e as ferramentas de depuração geralmente permitem que você faça alterações temporárias para poder continuar executando o programa.

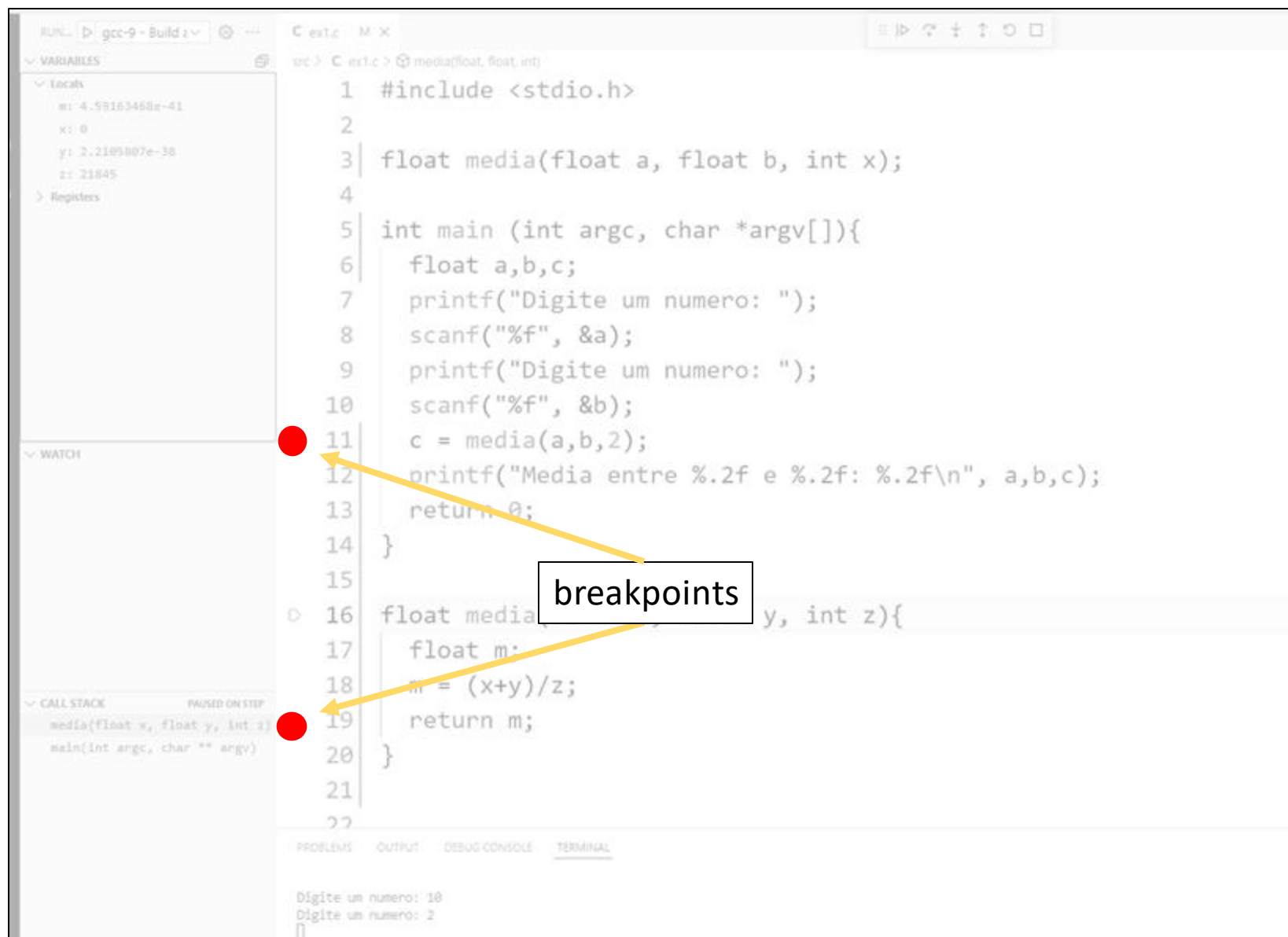
Usar um depurador com eficiência é também uma habilidade que leva tempo e prática, mas é definitivamente uma tarefa fundamental para todo desenvolvedor de software. Neste artigo, introduziremos os princípios fundamentais da depuração e forneceremos dicas para você começar.

[Como depurar para iniciantes absolutos](#)

DEPURAÇÃO DE CÓDIGO



DEPURAÇÃO DE CÓDIGO



DEPURAÇÃO DE CÓDIGO

The screenshot shows a C program being debugged. The code is as follows:

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

The left sidebar shows the **VARIABLES** panel with the following values:

- Locals
 - m: 4.59163468e-41
 - x: 0
 - y: 2.2105807e-38
 - z: 21845
- Registers

The **WATCH** panel is empty.

The **CALL STACK** panel shows the following stack:

- media(float x, float y, int z)
- main(int argc, char ** argv)

The **TERMINAL** panel shows the following output:

```
Digite um numero: 10
Digite um numero: 2

```

An arrow points from the text box to the **Run** button in the toolbar.

Executa o código até um breakpoint

DEPURAÇÃO DE CÓDIGO

The image shows a code editor with a debugger. The code is as follows:

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

The debugger interface includes a 'VARIABLES' pane on the left showing local variables: `m: 4.59163468e-41`, `x: 0`, `y: 2.2105807e-38`, and `z: 21845`. Below it is a 'WATCH' pane. At the bottom left is a 'CALL STACK' pane showing the current call stack: `media(float x, float y, int z)` and `main(int argc, char ** argv)`. The 'DEBUG CONSOLE' at the bottom shows the output: `Digite um numero: 10` and `Digite um numero: 2`.

Annotations highlight two debugger buttons:

- Executa o código até um breakpoint** (Run to Cursor): Indicated by an orange arrow pointing to the `Run to Cursor` button (a blue arrow pointing to the right).
- Pula a execução de um breakpoint** (Step Over): Indicated by a green arrow pointing to the `Step Over` button (a blue arrow pointing down).

DEPURAÇÃO DE CÓDIGO

Executa o código até um breakpoint

Pula a execução de um breakpoint

Entra em uma sub-rotina marcada por um breakpoint

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

VARIABLES

Locals

- x: 0
- y: 2.2105807e-38
- z: 21845

Registers

WATCH

CALL STACK

PAUSED ON STEP

- media(float x, float y, int z)
- main(int argc, char ** argv)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Digite um numero: 10
Digite um numero: 2
[]

DEPURAÇÃO DE CÓDIGO

Executa o código até um breakpoint

Pula a execução de um breakpoint

Entra em uma sub-rotina marcada por um breakpoint

Sai de uma sub-rotina marcada por um breakpoint

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Digite um numero: 10
Digite um numero: 2
[]

DEPURAÇÃO DE CÓDIGO

Executa o código até um breakpoint

Pula a execução de um breakpoint

Entra em uma sub-rotina marcada por um breakpoint

Sai de uma sub-rotina marcada por um breakpoint

Reinicia debug

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

Variables: m: 4.59163468e-41, x: 0, y: 2.2105807e-38, z: 21845

Call Stack: media(float x, float y, int z), main(int argc, char ** argv)

Terminal: Digite um numero: 10, Digite um numero: 2

DEPURAÇÃO DE CÓDIGO

The image shows a code editor window with a C program. The program defines a function `media` and a `main` function. The `main` function prompts the user for two numbers and calls `media`. The `media` function calculates the average of two numbers and a divisor. The code is as follows:

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

On the right side of the code editor, there are six colored boxes with arrows pointing to specific debug controls in the toolbar:

- Executa o código até um breakpoint** (Yellow box, points to the Run button)
- Pula a execução de um breakpoint** (Green box, points to the Step Over button)
- Entra em uma sub-rotina marcada por um breakpoint** (Blue box, points to the Step Into button)
- Sai de uma sub-rotina marcada por um breakpoint** (Red box, points to the Step Out button)
- Reinicia debug** (Purple box, points to the Restart button)
- Finaliza debug** (Pink box, points to the End button)

The left sidebar shows the **VARIABLES** panel with local variables `m`, `x`, `y`, and `z`. The **WATCH** panel is empty. The **CALL STACK** panel shows the current call stack with `media(float x, float y, int z)` and `main(int argc, char ** argv)`. The **TERMINAL** panel shows the output of the program:

```
Digite um numero: 10
Digite um numero: 2

```

DEPURAÇÃO DE CÓDIGO

The image shows a code editor with a debugger interface. The code is in C, showing a main function and a media function. The debugger is paused on line 16 of the media function. Various icons in the toolbar are highlighted with colored arrows pointing to text boxes explaining their functions.

Executa o código até um breakpoint (Yellow box)

Pula a execução de um breakpoint (Green box)

Entra em uma sub-rotina marcada por um breakpoint (Blue box)

Sai de uma sub-rotina marcada por um breakpoint (Red box)

Reinicia debug (Purple box)

Finaliza debug (Pink box)

Indica a linha que está sendo executada (Orange box)

```
1 #include <stdio.h>
2
3 float media(float a, float b, int x);
4
5 int main (int argc, char *argv[]){
6     float a,b,c;
7     printf("Digite um numero: ");
8     scanf("%f", &a);
9     printf("Digite um numero: ");
10    scanf("%f", &b);
11    c = media(a,b,2);
12    printf("Media entre %.2f e %.2f: %.2f\n", a,b,c);
13    return 0;
14 }
15
16 float media(float x, float y, int z){
17     float m;
18     m = (x+y)/z;
19     return m;
20 }
21
22
```

Variables: m: 4.59163468e-41, x: 0, y: 2.2105807e-38, z: 21845

Call Stack: media(float x, float y, int z), main(int argc, char ** argv)

Terminal: Digite um numero: 10, Digite um numero: 2

LISTA DE EXERCÍCIOS

- ❑ Abra a pasta Downloads no VS Code
 - ❑ Utilize o comando `git clone https://github.com/ECM404/lista3.git --recursive`
 - ❑ Entre na pasta utilizando o comando `cd lista3`
 - ❑ Inicialize o diretório com o comando `make install`