

SSC0501 - Introdução à Ciência de Computação I

Bem Vindos!

Depuração de Códigos (Debug!)

Depuração (Debug)

Depurar é como investigar: você precisa encontrar onde as coisas deram errado.

```
int matrix[3][3]:
```

Depuração é o processo de:

- Identificar erros em um programa.
- Compreender as causas desses erros.
- Corrigir o código para que ele funcione corretamente.

Tipos de Erros em Programas

Existem pelo menos três tipos de erros:

```
int matrix[3][3]:
```



5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	8

2	2	0	8	6	1
4	9	6	5	3	0
1	2	8	5	2	7

Tipos de Erros em Programas

Existem pelo menos três tipos de erros:

Tipo de Erro	Exemplo	Detectado Quando?
Erro de Sintaxe	Esqueceu um ;	Durante a compilação
Erro de Execução	Divisão por zero, ponteiro nulo	Durante a execução
Erro Lógico	Algoritmo errado, resultado incorreto	Difícil de detectar

Erro de Sintaxe

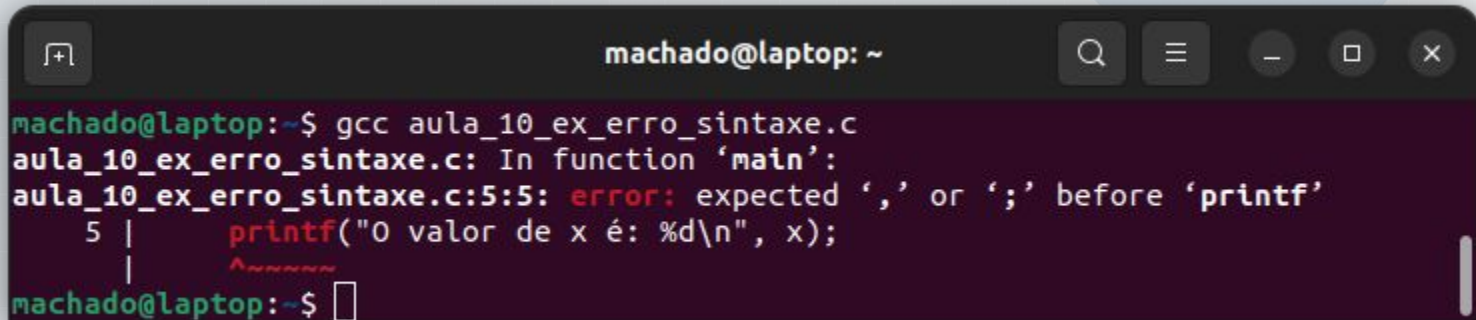
Exemplo: [ex_erro_sintaxe.c](#)

```
1 #include <stdio.h>
2
3 int main() {
4     int x = 10
5     printf("O valor de x é: %d\n", x);
6     return 0;
7 }
```

Erro de Sintaxe

Exemplo: [ex_erro_sintaxe.c](#)

```
1 #include <stdio.h>
2
3 int main() {
4     int x = 10
5     printf("O valor de x é: %d\n", x);
6     return 0;
7 }
```



A terminal window titled 'machado@laptop: ~' showing the compilation of a C program. The command 'gcc aula_10_ex_erro_sintaxe.c' is executed. The output shows a syntax error in the function 'main' at line 5, column 5: 'error: expected ',', or ';' before 'printf''. The error points to the line 'printf("O valor de x é: %d\n", x);' in the code snippet above.

```
machado@laptop:~$ gcc aula_10_ex_erro_sintaxe.c
aula_10_ex_erro_sintaxe.c: In function 'main':
aula_10_ex_erro_sintaxe.c:5:5: error: expected ',', or ';' before 'printf'
  5 |     printf("O valor de x é: %d\n", x);
    |     ^
machado@laptop:~$
```

Erro de Execução

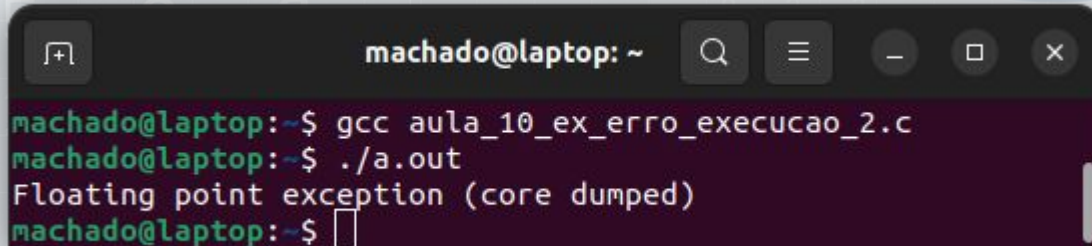
Exemplo: [aula_10_ex_erro_execucao.c](#)

```
1 #include <stdio.h>
2
3 int main() {
4     int a = 10;
5     int b = 0;
6     int resultado = a / b;
7     printf("Resultado: %d\n", resultado);
8     return 0;
9 }
```


Erro de Execução

Exemplo: [aula_10_ex_erro_execucao.c](#)

```
1 #include <stdio.h>
2
3 int main() {
4     int a = 10;
5     int b = 0;
6     int resultado = a / b;
7     printf("Resultado: %d\n", resultado);
8     return 0;
9 }
```



```
machado@laptop: ~
machado@laptop:~$ gcc aula_10_ex_erro_execucao_2.c
machado@laptop:~$ ./a.out
Floating point exception (core dumped)
machado@laptop:~$
```

Erro de Lógico

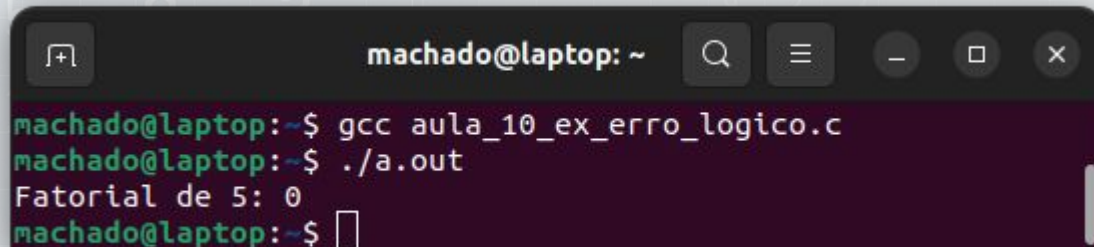
Exemplo: [aula_10_ex_erro_logico.c](#)

```
1 #include <stdio.h>
2
3 // Objetivo: calcular o fatorial de 5
4
5 int main() {
6     int fatorial = 0;
7     for (int i = 1; i <= 5; i++) {
8         fatorial = fatorial * i;
9     }
10    printf("Fatorial de 5: %d\n", fatorial);
11    return 0;
12 }
```

Erro de Lógico

Exemplo: [aula_10_ex_erro_logico.c](#)

```
1 #include <stdio.h>
2
3 // Objetivo: calcular o fatorial de 5
4
5 int main() {
6     int fatorial = 0;
7     for (int i = 1; i <= 5; i++) {
8         fatorial = fatorial * i;
9     }
10    printf("Fatorial de 5: %d\n", fatorial);
11    return 0;
12 }
```



```
machado@laptop: ~
machado@laptop:~$ gcc aula_10_ex_erro_logico.c
machado@laptop:~$ ./a.out
Fatorial de 5: 0
machado@laptop:~$
```

Por que Depurar?

- Programas raramente funcionam perfeitamente na primeira tentativa.
- Depuração ajuda a **entender melhor o código**.
- Permite encontrar **bugs difíceis de perceber**.
- Melhora a qualidade e robustez do software.

Métodos de Depuração

```
int matrix[3][3]:
```


Métodos de Depuração

- **Depuração manual** (ex: printf)
- **Depuradores interativos** (ex: gdb)
- **Testes automatizados**
- **Ferramentas de análise de memória** (ex: valgrind)

Boas Práticas

- Teste seu código com entradas simples primeiro.
- Isolar o problema: comente partes suspeitas.
- Leia com atenção os avisos do compilador.
- Sempre compile com -Wall.



5	9	3
2	6	8
4	0	7

```
int matrix[3][3]:
```

5	9	3
2	6	8
4	0	7

9	2	3
7	8	5
2	2	3

2	2	0	8	6	1
4	9	6	5	5	3
1	2	8	5	2	

Depuração manual

GNU Debugger (gdb)

O gdb (Depurador Interativo)

O GNU Debugger (gdb) é o depurador padrão para programas em C, C++, Fortran, entre outras linguagens. Ele permite:

- Executar o programa **linha por linha**.
- Parar em pontos específicos do código (breakpoints).
- Verificar e alterar **valores de variáveis**.
- Investigar **chamadas de função (stack trace)**.
- Verificar **fluxo de controle e desvios de execução**.
- Identificar **causas de falhas como segmentation faults**.

Integração com IDEs

- **VS Code** – através da extensão C/C++ da Microsoft
- **Code::Blocks** – suporte nativo ao GDB

Outras:

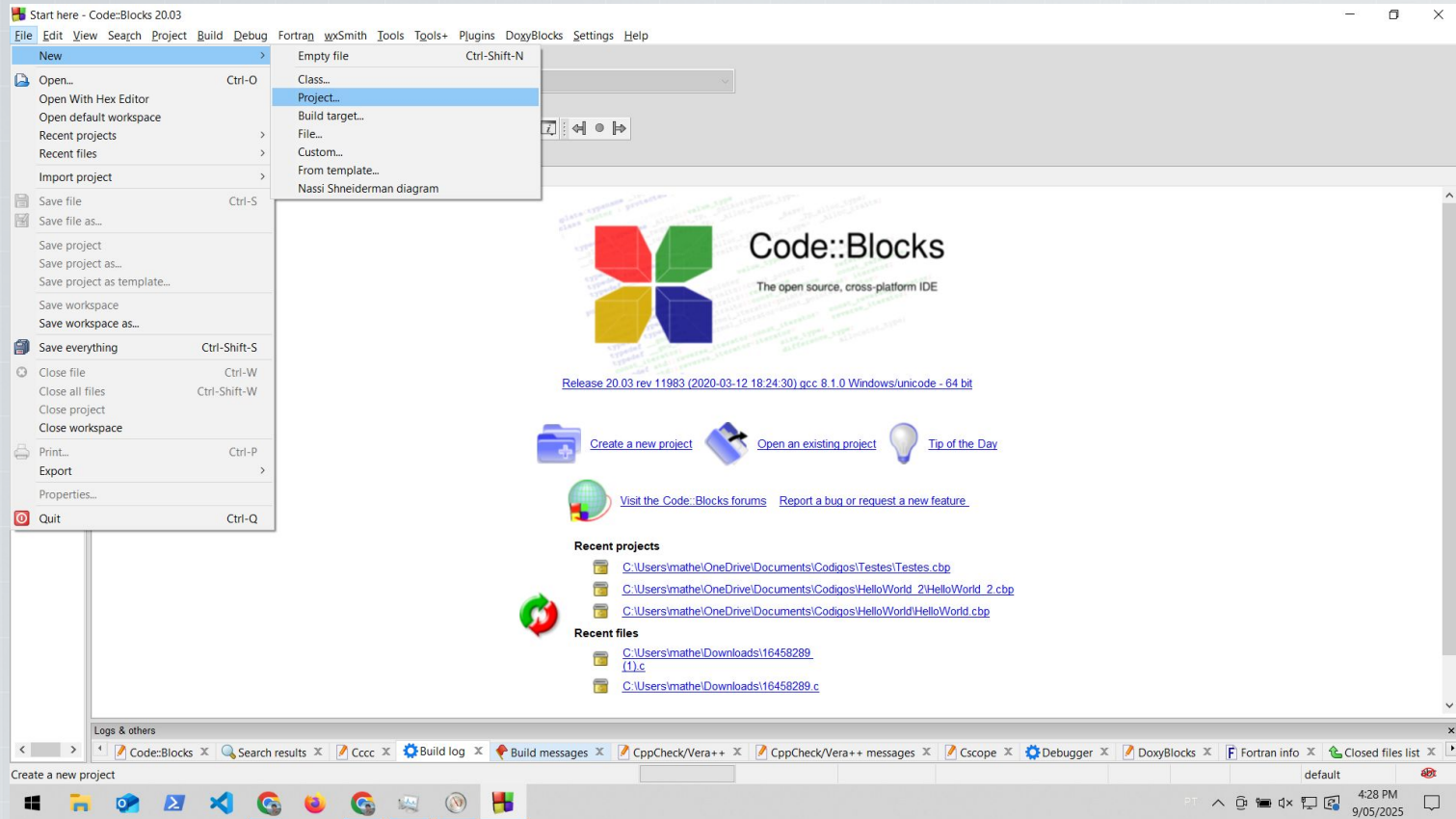
- CLion (JetBrains) – integração completa com GDB
- Eclipse CDT – suporte integrado ao GDB
- Dev-C++ – suporte básico ao GDB
- Qt Creator – integração embutida com GDB

Integração com **Code::Blocks**

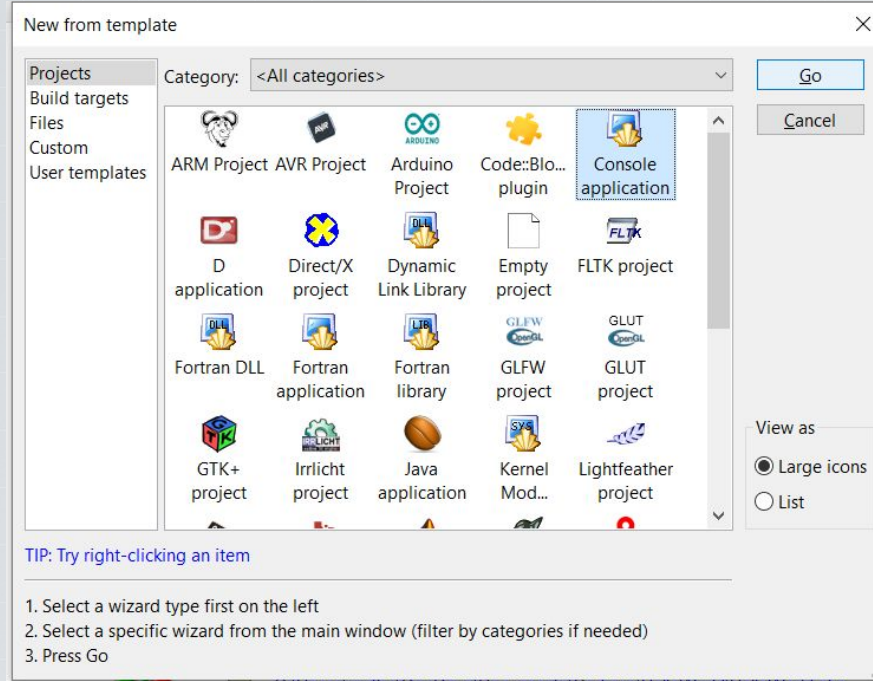
Pré-requisitos

- Instale o **Code::Blocks** com compilador **MinGW/GCC** (versão “with compiler”)
- Certifique-se de que o `gdb.exe` está instalado (vem junto com MinGW)
- Código deve estar **compilado com informações de depuração**:
`gcc -g arquivo.c -o programa.exe`
O **Code::Blocks** faz isso automaticamente se configurado corretamente

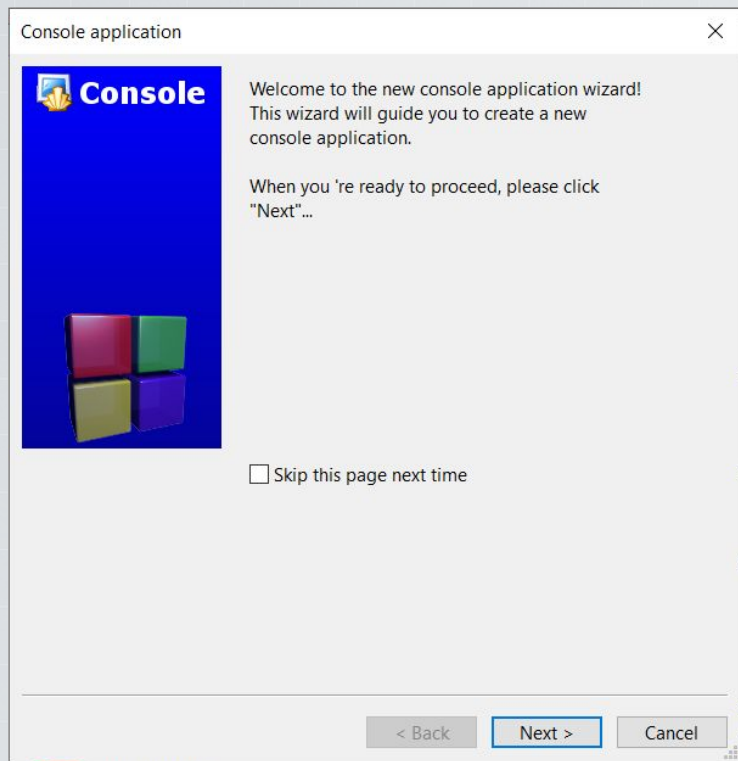
Criação de um novo projeto



Criação de um novo projeto

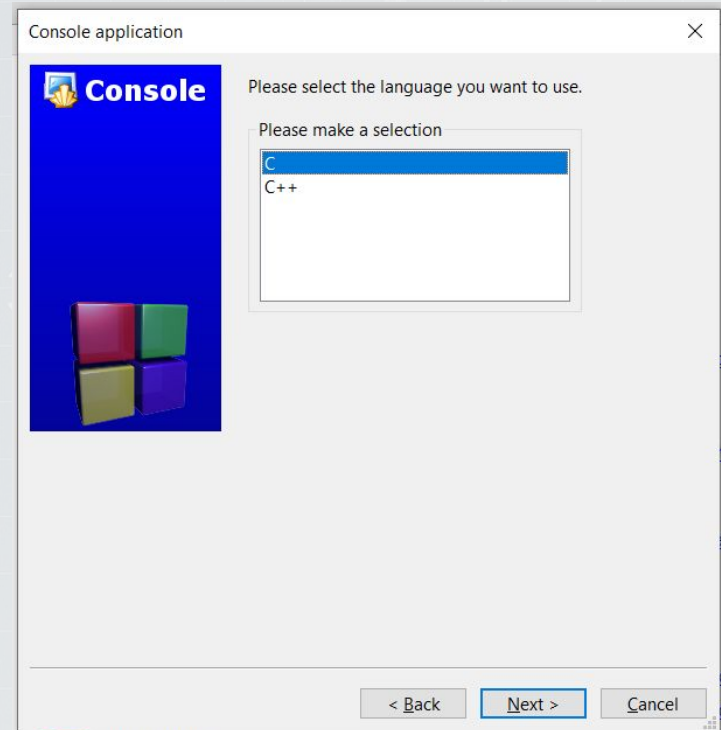


Configurando Criação do Projeto



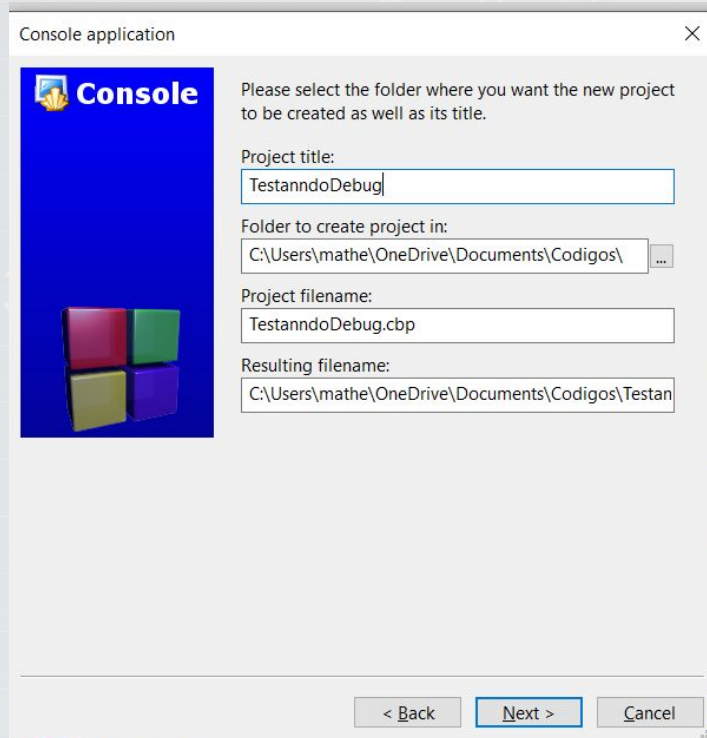
Configurando Criação do Projeto

- Selecione Linguagem C



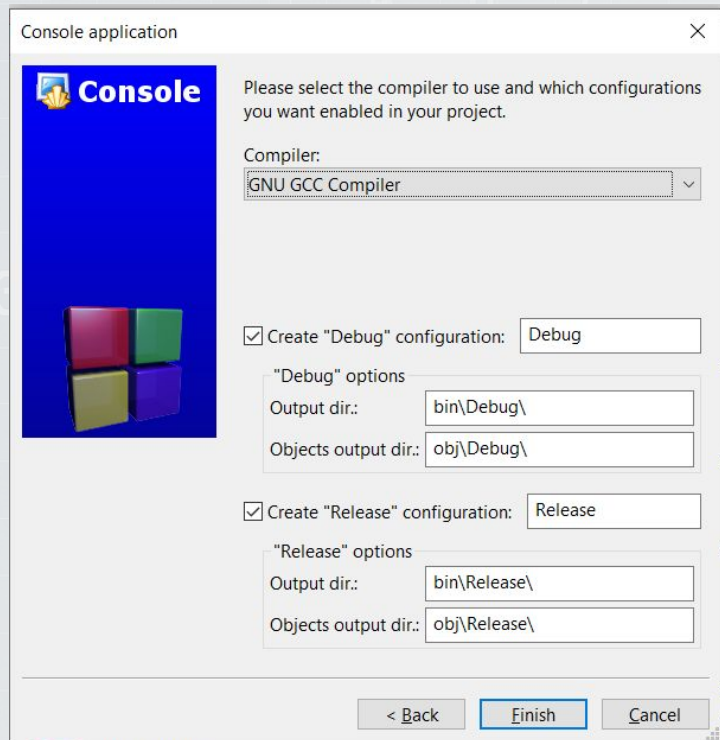
Configurando Criação do Projeto

- Digite o nome do projet
- Selecione a pasta para salvar o projeto

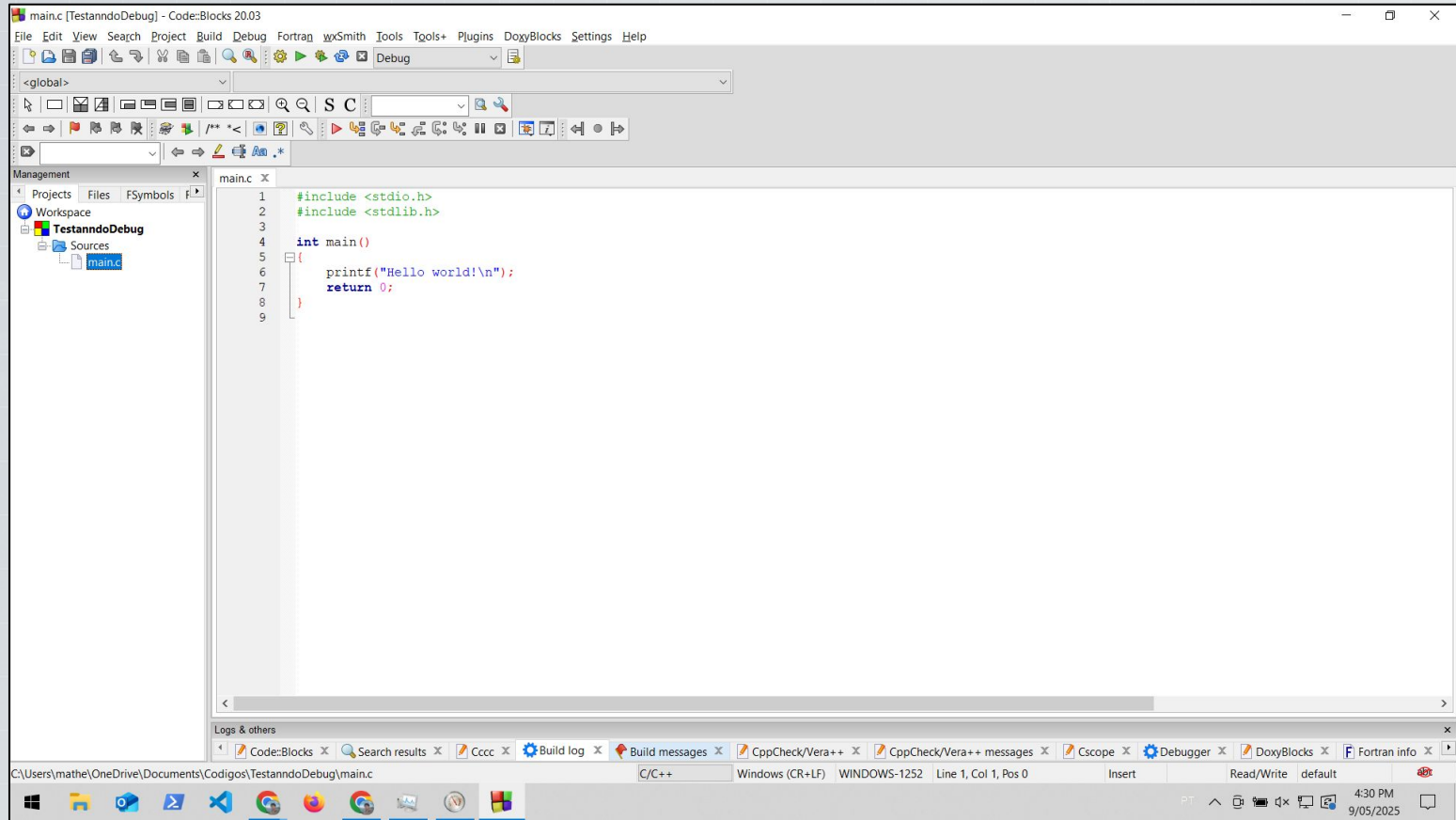


Configurando Criação do Projeto

- Importante marcar “Debug” para que o código seja compilado com -g e desta forma gere um binário com informações de depuração.

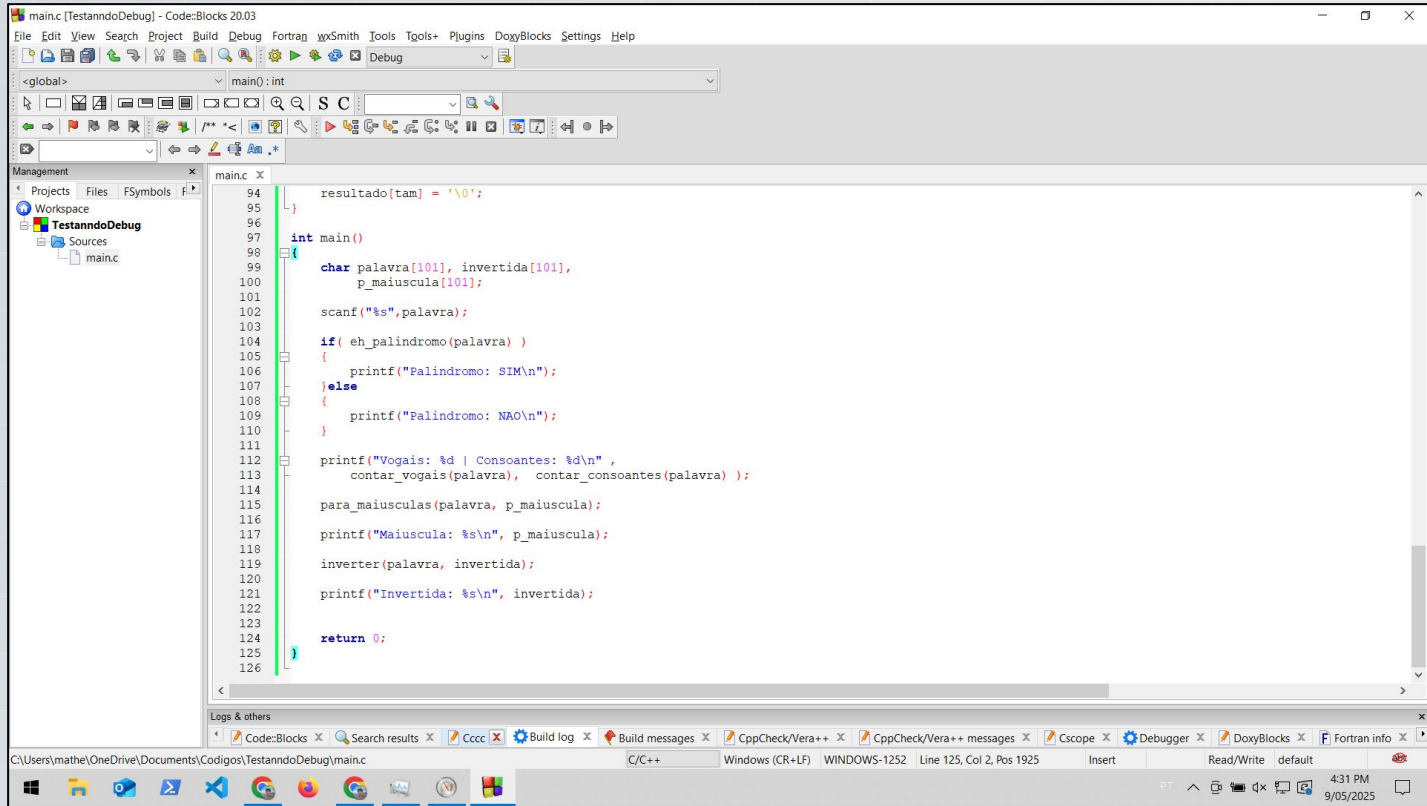


Projeto Padrão Criado

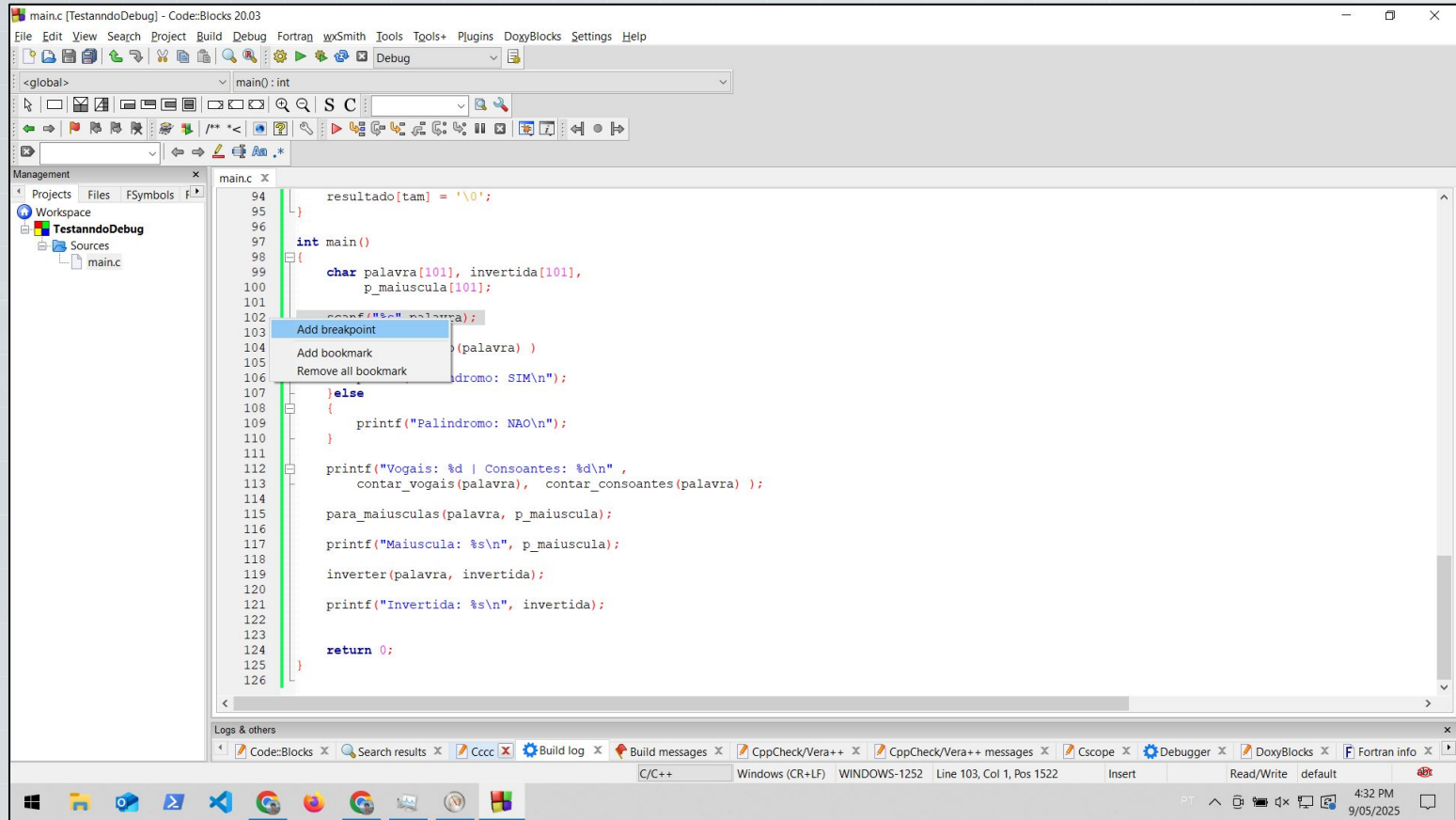


Carregando Código para depuração

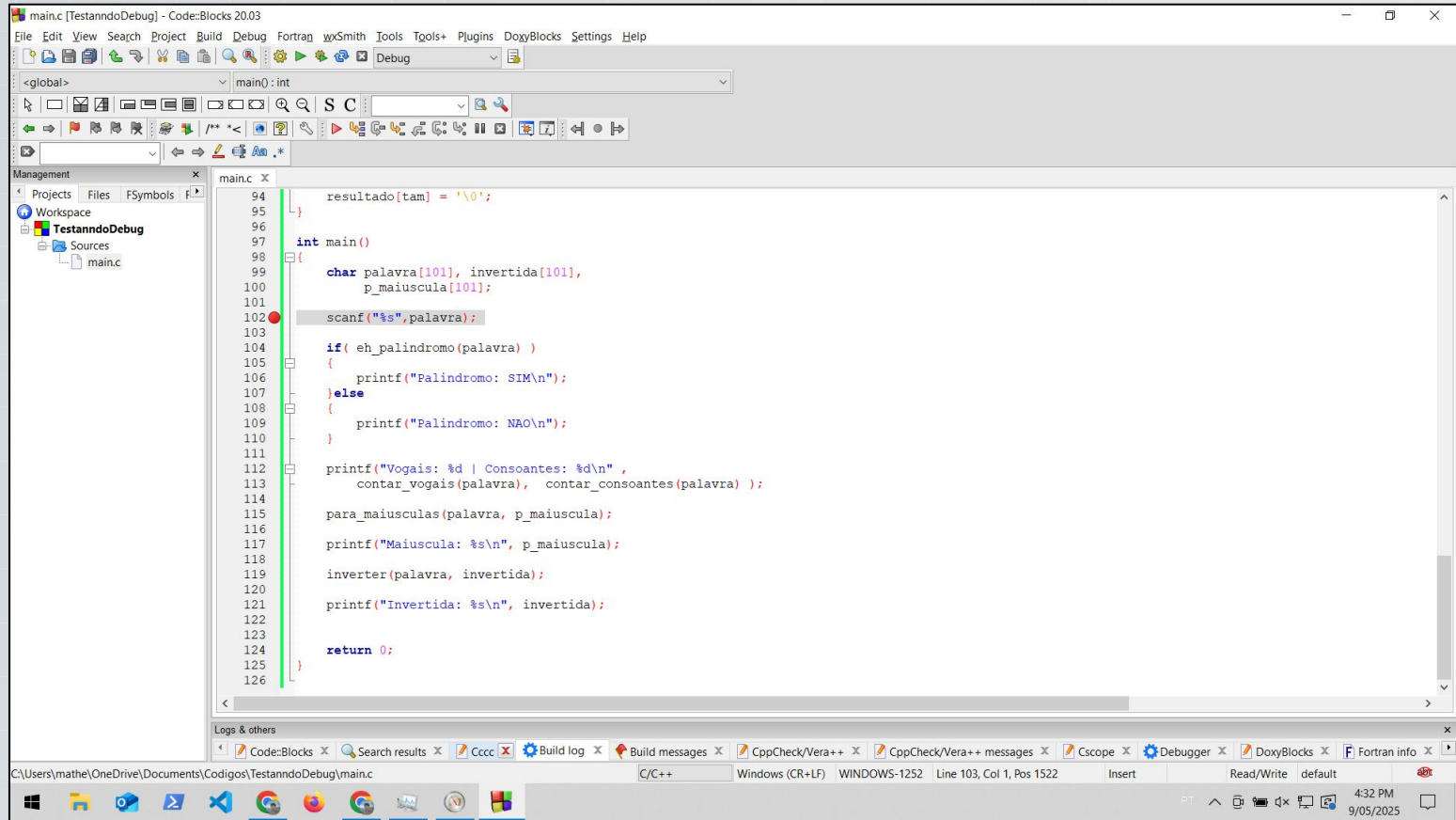
[analisa_palavra.c](#)



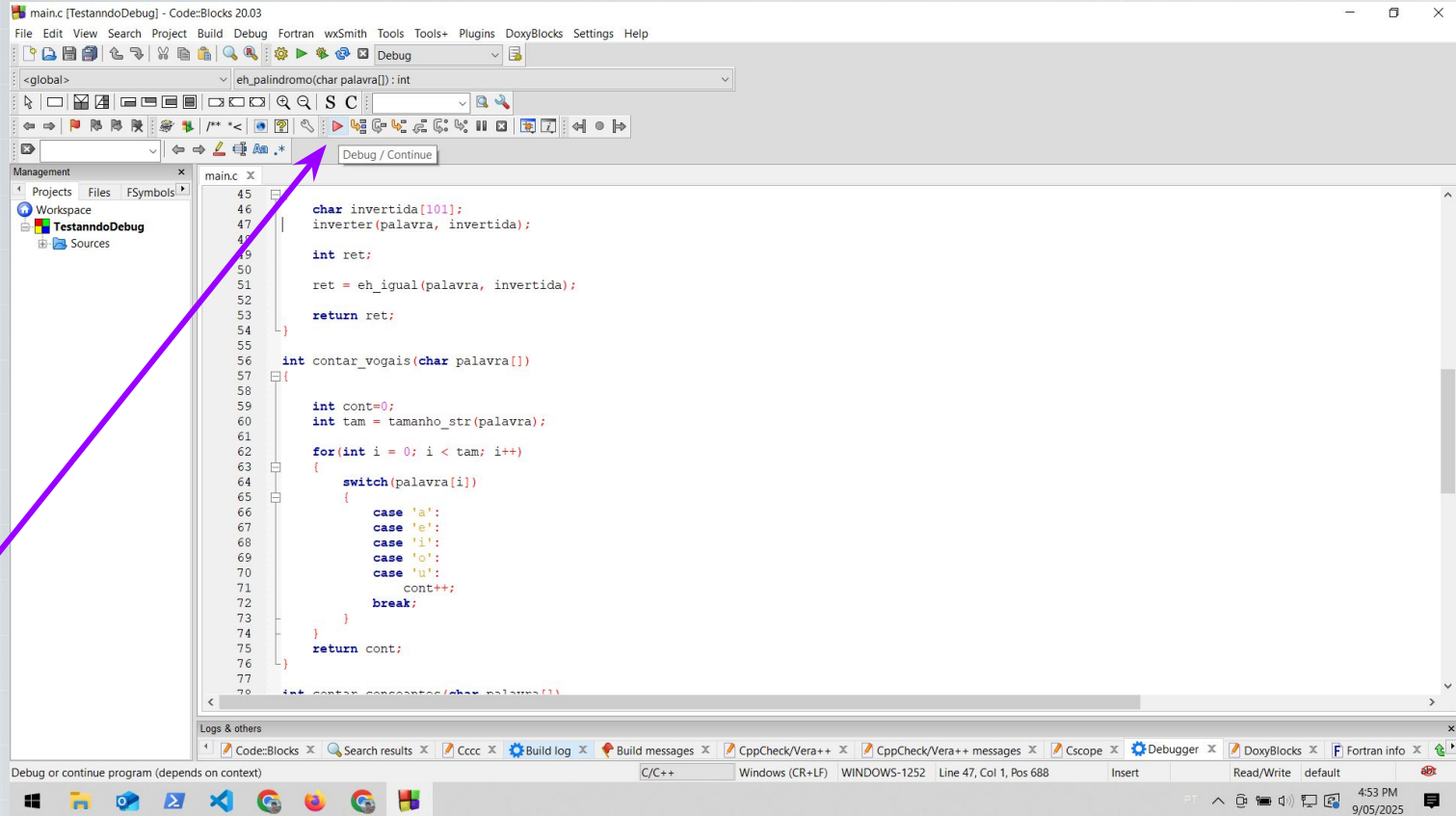
Adicionando Ponto de Parada (breakpoint)



Adicionando Ponto de Parada (breakpoint)



Iniciando Depuração



Testes Automatizados

Valgrind (Analizador de memoria)

Valgrind

- O Valgrind precisa estar instalado

\$ sudo apt install valgrind

- O programa precisa ser compilado com -g apenas para que o valgrind mostre partes do código com problemas

\$ valgrind --leak-check=full ./programa

Prática!

Código Matricula: R4SM

<https://runcodes.icmc.usp.br/offerings/view/83>

[run.codes]



Menu Professor ▾

matheus.m.santos@icmc.usp.br ▾

Hora do Servidor: 14/03/2023 18:46:06

Home > SSC0501

SSC0501 - Introdução à Ciência de Computação I

Professores/Monitores

Professores: Matheus Machado dos Santos
Turma: 2025101
Universidade: USP
Ativa até: 21/07/2025



Código de Matrícula

R4SM



Novo Exercício



Enviar E-mail



Ver Notas



Exportar Tabela de Notas

Exercícios

No.	Exercício	Status	Casos Corretos	Nota	Entregas	Participantes	Prazo de Entrega	Ações
1	Hello World	Finalizado	1/1	10.00	78	43/46	14/03/2025 21:00:00	Ver Detalhes Remover Exercício
2	1.01 Maior número	Não Entregue	0/6	0	0	0/46	19/03/2025 23:59:59	Ver Detalhes Remover Exercício
3	1.02 Par ou ímpar	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59	Ver Detalhes Remover Exercício
4	1.03 Positivo, negativo ou zero	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59	Ver Detalhes Remover Exercício
5	1.04 Maior de três números	Não Entregue	0/7	0	0	0/46	19/03/2025 23:59:59	Ver Detalhes Remover Exercício

SSC0501 - Introdução à Ciência de Computação I

Obrigado pela atenção!!

Integração com VS Code

Passos VSCode

- Criar pasta do projeto
- Inserir código na pasta
- Abrir a pasta com VS Code

```
int matrix[3][3]:
```

Crando task de compilação

Criar o tasks.json (compilação)

- Pressione Ctrl+Shift+P para abrir a paleta de comandos.
- Digite e selecione “Tasks: Configure Default Build Task”
- Escolha “C/C++: gcc build active file” ou similar.
- O VS Code cria o arquivo .tasks.json na configuração base.
- Edite, se neces

```
vscode > {} tasks.json > [ ] tasks > {} 0
1 {
2   "version": "2.0.0",
3   "tasks": [
4     {
5       "type": "cppbuild",
6       "label": "C/C++: gcc build active file",
7       "command": "/usr/bin/gcc",
8       "args": [
9         "-fdiagnostics-color=always",
10        "-g",
11        "${file}",
12        "-o",
13        "${fileDirname}/${fileBasenameNoExtension}"
14      ],
15      "options": {
16        "cwd": "${fileDirname}"
17      },
18      "problemMatcher": [
19        "$gcc"
20      ],
21      "group": {
22        "kind": "build",
23        "isDefault": true
24      },
25      "detail": "compiler: /usr/bin/gcc"
26    ]
27  ]
28 }
```

Select the task to be used as the default build task

roscore: roscore
roslaunch: roslaunch
C/C++: clang-14 build active file
compiler: /usr/bin/clang-14
C/C++: gcc build active file
compiler: /usr/bin/gcc
C/C++: gcc-12 build active file
compiler: /usr/bin/gcc-12
C/C++: gcc-11 build active file
compiler: /usr/bin/gcc-11

Criar o launch.json (debug)

- Vá para a aba Executar e Depurar no menu lateral esquerdo (ou Ctrl+Shift+D).
- Clique em “Criar um arquivo launch.json”.
- Escolha “C++ (GDB/LLDB)”.
- Selecione “gcc build and debug active file”.
- O VS Code criará o launch.json com base no arquivo .c aberto.

