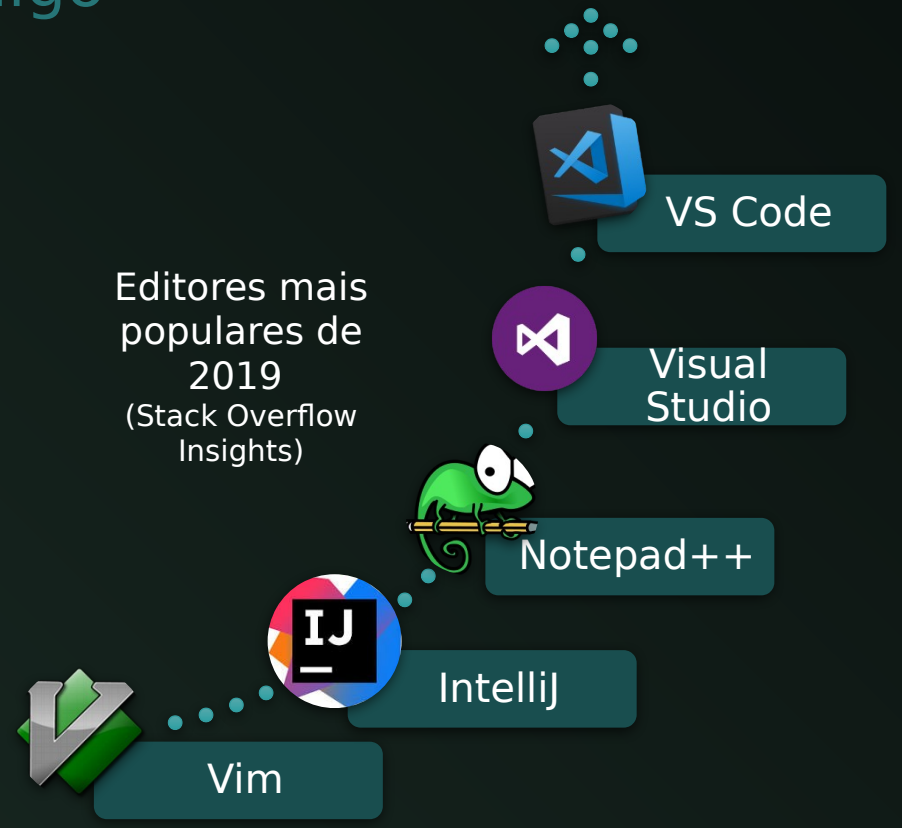
Judson Santos Santiago

Ambiente de Trabalho

Compiladores

Introdução

- O Visual Studio Code é um editor de código
  - Gratuito
  - Código aberto
  - Cross-plataforma (Windows, Linux e MacOS)
- Instalação:
  - Baixar arquivo .deb (Debian, Ubuntu, Mint): https://code.visualstudio.com/
  - Instruções para instalação no Linux: https://code.visualstudio.com/docs/setup/linux

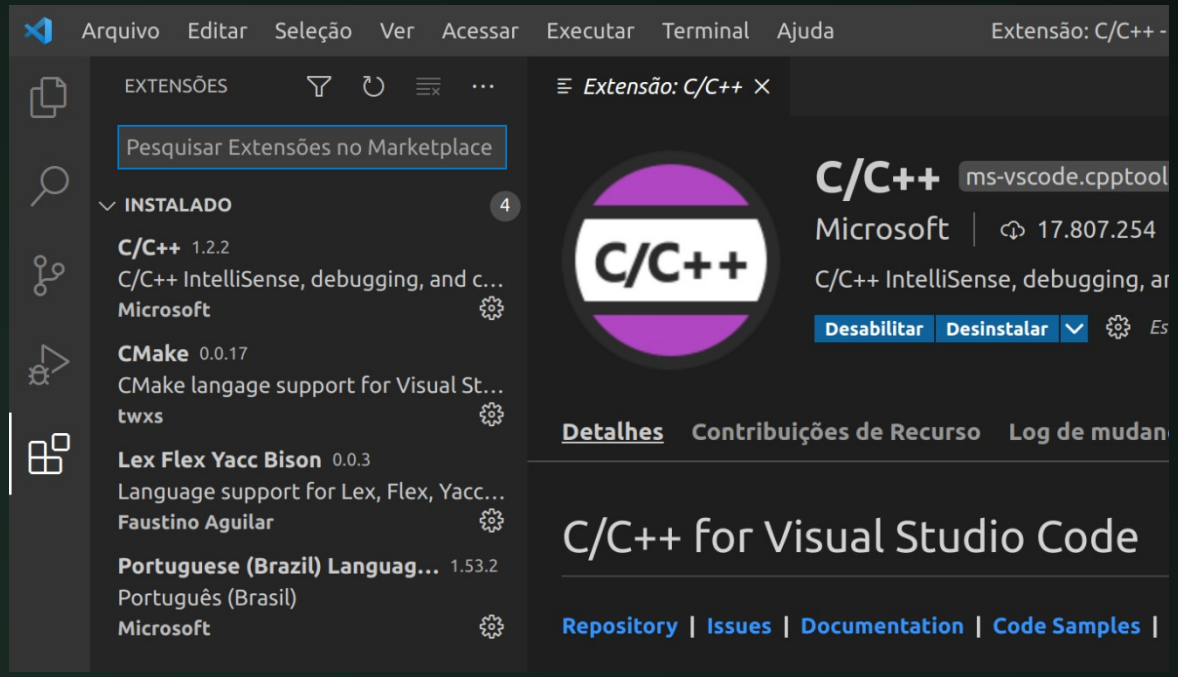


Visual Studio Code

- O Visual Studio Code não é uma IDE completa e não possui:
  - Compilador
  - Ferramentas de depuração
  - Sistema de compilação
- Mas é configurável:
  - Pode trabalhar com ferramentas externas:
    - Incluindo g++, gdb, make e cmake
  - Possui suporte a extensões (plugins):
    - Possui loja dentro do próprio editor

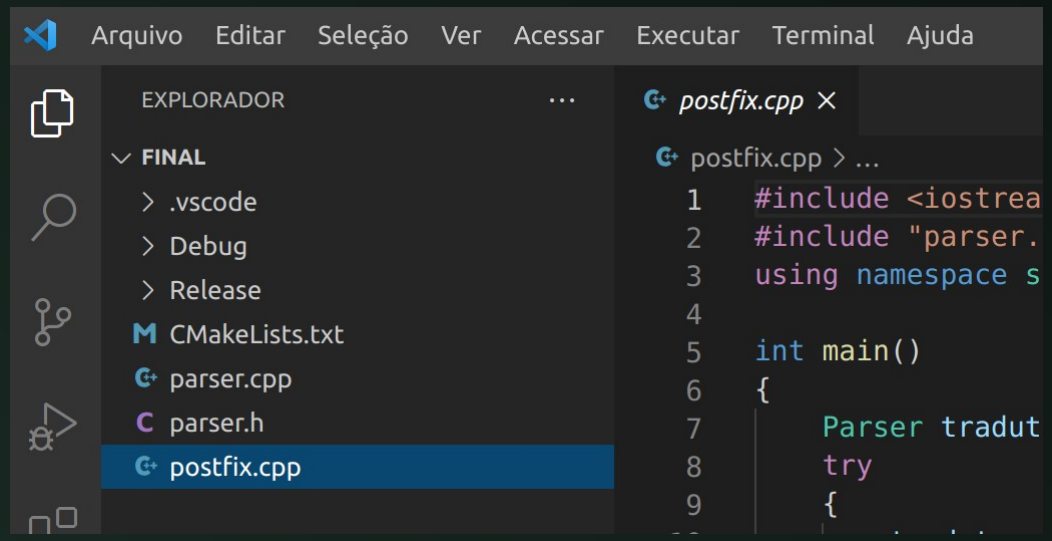
Extensões

- Para a disciplina, instalar as seguintes extensões:
  - C/C++
    Microsoft
  - CMake
    twxs
  - Lex Flex Yacc Bison
    Faustino Aguilar
  - Portuguese Brazil Language Pack
    Microsoft



Projetos e Pastas

- Abrindo Pastas
  - Menu Arquivo - Abrir Pasta (Ctrl+K Ctrl+O)
    - Permite abrir um projeto criado anteriormente
    - Possibilita criação de tarefas:
      - Compilação
      - Depuração
  - Abrir um arquivo permite apenas a edição de código
  - Para compilar e depurar é preciso abrir uma pasta



Compilação

- A compilação requer configuração de uma tarefa
  - Menu Terminal – Configurar Tarefas...
  - Criar arquivo tasks.json do modelo - Others

tasks.json

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build",
      "type": "shell",
      "command": "g++ parser.cpp postfix.cpp -g -std=c++17 -o
postfix"
    }
  ]
}
```

Realiza a compilação direta, sem usar um sistema de compilação

Compilação

- Configurando outras opções da tarefa:
  - Filtro de Problemas: exibe os erros de compilação na aba "Problemas"

```
    "problemMatcher": "$gcc",
```

  - Tarefa padrão de compilação: torna essa a tarefa padrão para a compilação, permitindo seu acionamento direto através:
    - Menu Terminal – Executar Tarefa de Compilação (Ctrl+Shift+B)

```
    "group": {
          "kind": "build",
          "isDefault": true
      },
```

Compilação

- Configurando outras opções da tarefa:
  - Apresentação no painel:
    - `"reveal": "always"` - sempre abre o painel na compilação
    - `"focus": true` - muda o foco da janela para o painel
    - `"panel": "new"` - sempre abre um painel novo e limpo

```
    "presentation": {
        "echo": true,
        "reveal": "always",
        "focus": true,
        "panel": "shared",
        "showReuseMessage": true,
        "clear": true
    }
```

Depuração

- A depuração requer a configuração de launch.json
  - Executar – Iniciar Depuração (F5) - C++ (GDB/LLDB) – Configuração Padrão

launch.json

```
...
"program": "${workspaceFolder}/postfix",
"args": [],
"stopAtEntry": false,
...
```

Adicione um breakpoint ou mude a opção `stopAtEntry` para entrar em modo de depuração passo a passo.

- `"preLaunchTask": "build"` da depuração

Usando Make

- O GNU Make pode ser usado através de uma tarefa
  - Em vez de chamar diretamente o compilador:

```
    "command": "g++ parser.cpp postfix.cpp -g -std=c++17 -o
postfix"
```

  - Pode-se compilar usando um makefile previamente criado:
    - Manualmente
    - CMake

```
    "command": "make"
```

Usando CMake

- Tarefa para gerar um makefile usando CMake

tasks.json

```
"tasks": [
    {
        "label": "cmake",
        "type": "shell",
        "command": "cmake",
        "args": ["../"],
        "group": "build",
        "options": {
            "cwd": "${workspaceFolder}/Debug",
        },
        "problemMatcher": [ ],
    },
]
```

Argumentos podem ser passados para os comandos

É necessário definir a pasta de trabalho, local onde será executado o comando cmake

CMake Debug e Release

- Um arquivo tasks.json pode conter várias tarefas

```
{
    "label": "cmake debug",
    "type": "shell",
    "command": "cmake",
    "args": [
        "-DCMAKE_BUILD_TYPE=Debug",
        "../"
    ],
    "group": "build",
    "options": {
        "cwd":
"${workspaceFolder}/Debug",
    },
    "problemMatcher": [ ],
},
```

```
{
    "label": "cmake release",
    "type": "shell",
    "command": "cmake",
    "args": [
        "-DCMAKE_BUILD_TYPE=Release",
        "../"
    ],
    "group": "build",
    "options": {
        "cwd":
"${workspaceFolder}/Release",
    },
    "problemMatcher": [ ],
},
```

Make Debug e Release

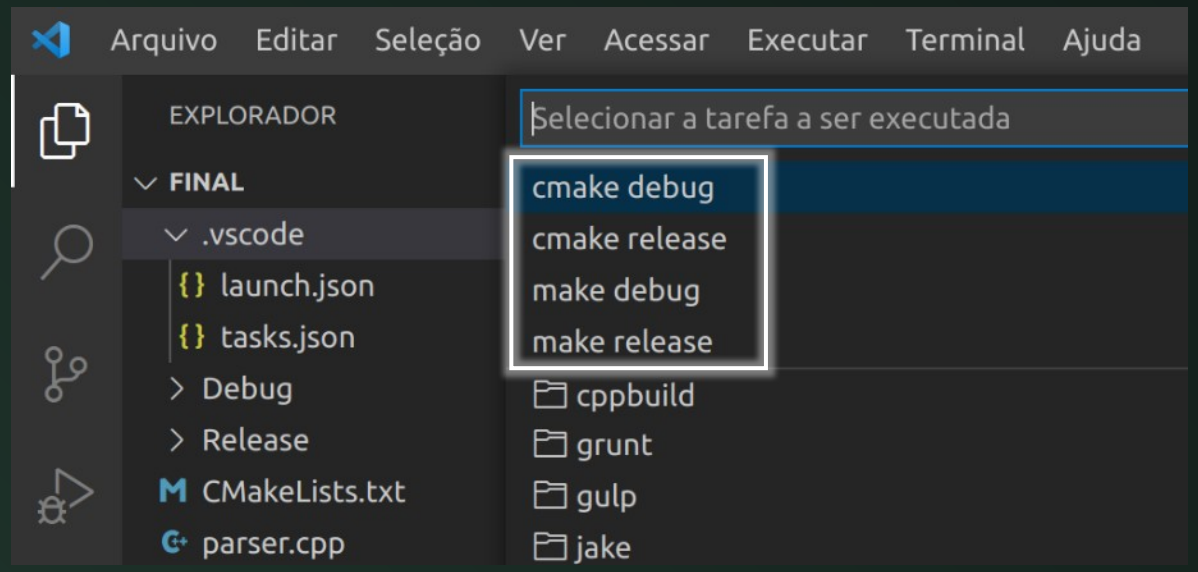
- Permite ter um Makefile para Debug e outro para Release

```
{
    "label": "make debug",
    "type": "shell",
    "command": "make",
    "args": [],
    "group": {
        "kind": "build",
        "isDefault": true
    },
    "options": {
        "cwd":
"${workspaceFolder}/Debug",
    },
    "problemMatcher": ["$gcc"],
},
```

```
{
    "label": "make release",
    "type": "shell",
    "command": "make",
    "args": [],
    "group": "build",
    "options": {
        "cwd":
"${workspaceFolder}/Release",
    },
    "problemMatcher": ["$gcc"],
},
```

Tarefas

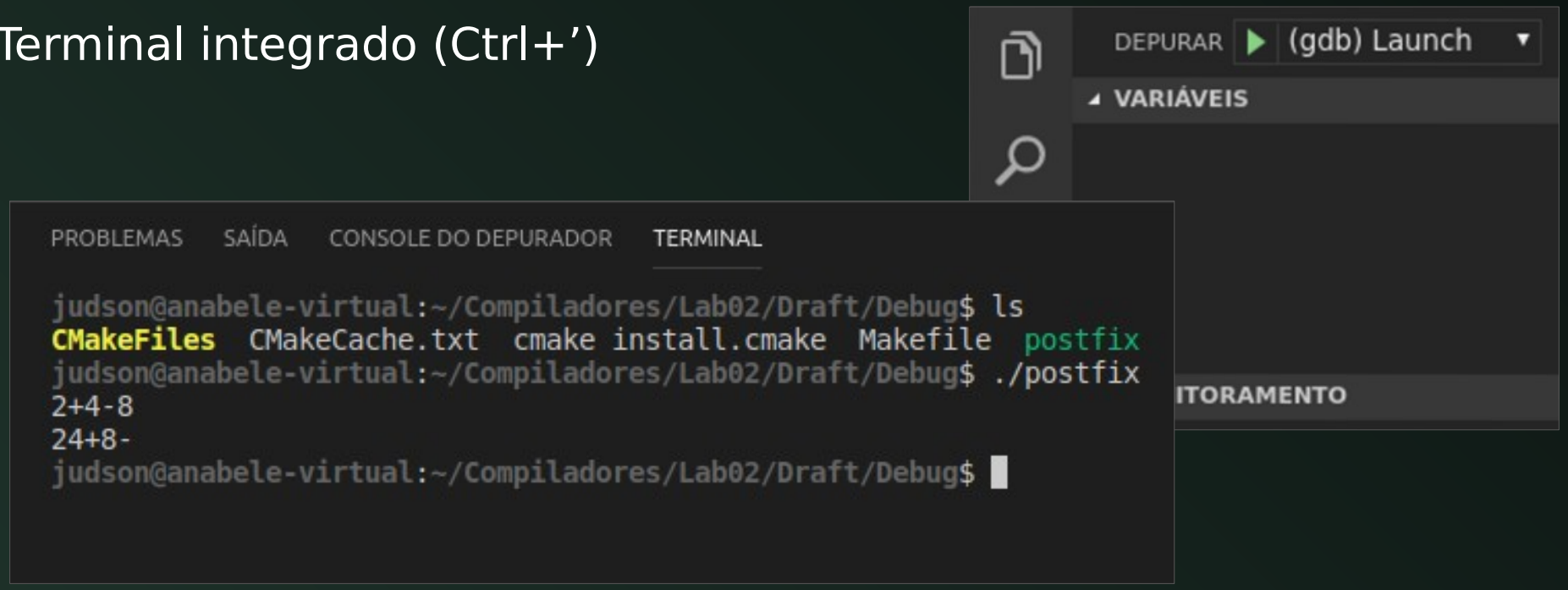
- As tarefas podem ser executadas através:
  - Menu Terminal - Executar Tarefa...
  - Ctrl+Shift+B para a tarefa padrão de compilação



Tarefas configuradas

Rodando o Programa

- A execução do programa pode ser feita via
  - Depurador
  - Terminal integrado (Ctrl+’)



Resumo

- Vamos trabalhar no Linux
  - Usando o terminal
    - ls, mkdir, cd, rm, cat, <, >, |, etc.
  - Com a linguagem C++
    - g++, gdb, make, cmake
  - Editando código no Visual Studio Code
    - Vira uma IDE com a configuração apropriada
      - Compilação: arquivo tasks.json
      - Depuração: arquivo launch.json