Compilação e Depuração com GCC/GDB e VS Code

Introdução às Técnicas de Programação

Prof. Fernando Figueira

Setembro 2025

Agenda

1. Revisão: Processo de Compilação
2. Depuração com GDB
3. Depuração no VS Code
4. Demonstração Prática

O que é Compilação?

- **Compilação**: Processo de transformar código-fonte em código executável
- **GCC**: GNU Compiler Collection - compilador gratuito e amplamente usado
- **Etapas**:
  1. Pré-processamento (diretivas `#include`, `#define`)
  2. Compilação (código C $\rightarrow$ assembly)
  3. Montagem (assembly $\rightarrow$ código objeto)
  4. Ligação (código objeto $\rightarrow$ executável)

```
1 gcc arquivo.c -o programa
```

Listing 1: Comando básico de compilação

Compilação Básica

Comandos essenciais

```
# Compilacao simples
gcc divide_outro.c -o divide_outro

# Execucao
./divide_outro

# Compilacao com avisos (recomendado)
gcc -Wall divide_outro.c -o divide_outro

# Compilacao para debug
gcc -g -Wall divide_outro.c -o divide_outro
```

- `-Wall`: Habilita avisos do compilador
- `-g`: Inclui informações de debug no executável
- `-o`: Especifica o nome do arquivo de saída

O que é Depuração?

- **Depuração**: Processo de encontrar e corrigir erros (bugs) no código
- **GDB**: GNU Debugger - ferramenta de linha de comando para depuração
- **Quando usar?**:
  - Programa trava ou gera erro
  - Resultado incorreto
  - Entender fluxo de execução
  - Verificar valores de variáveis durante execução

Pré-requisito

Compilar com flag -g para incluir informações de debug

GDB: Comandos Básicos

Iniciando o GDB

```
# Compilar com debug
gcc -g -Wall divide_outro.c -o divide_outro

# Iniciar GDB
gdb ./divide_outro
```

Comandos principais:

- `run` (r) - Executar
- `break` (b) - Breakpoint
- `step` (s) - Próxima linha
- `next` (n) - Próxima linha (sem entrar em funções)
- `continue` (c) - Continuar

Inspeção:

- `print` (p) - Valor de variável
- `info locals` - Todas as variáveis locais
- `list` (l) - Ver código
- `quit` (q) - Sair

GDB: Exemplo Prático

Sessão de debug paso-a-passo

```
$ gdb ./divide_outro
(gdb) break main          # Breakpoint na funcao main
(gdb) run                 # Executar programa
(gdb) list                # Ver codigo atual
(gdb) break 11            # Breakpoint na linha 11
(gdb) continue            # Continuar ate breakpoint
12 24                     # Entrada: digitar dois numeros
(gdb) print a             # Ver valor de 'a'
(gdb) print b             # Ver valor de 'b'
(gdb) step               # Executar proxima linha
(gdb) print resto        # Ver valor de 'resto'
(gdb) continue           # Continuar execucao
```

VS Code: Configuração Inicial

- **Extensões necessárias**:
  - C/C++ (Microsoft)
  - C/C++ Extension Pack (opcional, mas recomendado)
- **Arquivos de configuração**:
  - `.vscode/tasks.json` - Tarefas de compilação
  - `.vscode/launch.json` - Configuração de debug

Dica

VS Code pode gerar essas configurações automaticamente quando você tenta debugar pela primeira vez

VS Code: Usando o Debugger

1. **Definir breakpoints**: Clicar na margem esquerda do editor (números das linhas)
2. **Iniciar debug**: F5 ou menu Debug → Start Debugging
3. **Controles durante debug**:
   - F10 - Step Over (próxima linha)
   - F11 - Step Into (entrar em função)
   - Shift+F11 - Step Out (sair de função)
   - F5 - Continue
   - Shift+F5 - Stop
4. **Painéis importantes**:
   - **Variables** - Valores das variáveis
   - **Watch** - Expressões para monitorar
   - **Call Stack** - Pilha de chamadas
   - **Terminal** - Onde digitar entrada do programa

IMPORTANTE: Entrada de Dados no VS Code

Onde digitar quando o programa pede entrada?

- **Durante debug**: Digite no painel **TERMINAL** (não no Debug Console)
- **Localização**: Parte inferior da tela, aba "TERMINAL"
- **Exemplo**: Quando `scanf("%d %d", &a, &b)` aguarda entrada

Fluxo típico

1. Programa executa até `scanf`
2. Execução pausa aguardando entrada
3. Clique na aba `TERMINAL`
4. Digite os valores (ex: `12 24`)
5. Pressione Enter
6. Programa continua

Vamos Praticar!

Cenário: Debugar divide_outro.c

Vamos encontrar e entender o comportamento do programa:

1. Compilar com GCC
2. Debugar via linha de comando (GDB)
3. Debugar no VS Code
4. Comparar as duas abordagens

Casos de teste

- Entrada: 12 4 (4 divide 12)
- Entrada: 7 3 (nenhum divide o outro)
- Entrada: 15 15 (números iguais)

Exercício para Casa

Prática individual

1. Crie um programa em C que calcule o fatorial de um número
2. Inclua pelo menos um erro intencional (ex: condição de parada incorreta)
3. Use GDB para encontrar e corrigir o erro
4. Repita o processo no VS Code
5. Documente as diferenças entre as duas abordagens

Entrega

Não é necessário entregar - é para fixação do conteúdo

Resumo

- **GCC**: Compilador essencial para C
- **Flag -g**: Fundamental para debug
- **GDB**: Poderoso debugger de linha de comando
- **VS Code**: Interface visual amigável para debug
- **Breakpoints**: Ferramenta essencial para inspeção
- **Entrada no VS Code**: Use o painel TERMINAL durante debug

Próxima aula

Estruturas de repetição (for, while, do-while)

Perguntas?

Dúvidas sobre compilação e depuração?

Obrigado!