

Organização de Computadores

Versão Hands-on com Logisim

Prof. Juan G. Colonna
juancolonna@icomp.ufam.edu.br
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)
Semestre 2024/01

Técnicas para melhorar o montador

Clean code

Regras principais:

1. A definição de nome é essencial para o bom entendimento de um código, seja: variável, função, classe, etc. Ao definir um nome, é preciso ter em mente 2 pontos principais:
 - a. Ele deve ser preciso e autoexplicativo;
 - b. Não se deve ter medo de nomes grandes (sem exagerar).



Clean code

Regras principais:

2. Não bagunçar.
 - a. Deixar o código mais limpo do que estava antes de mexer nele.



Clean code

Regras principais:

3. Seja o verdadeiro autor do código. Pense de forma narrativa, portanto, o código é uma história e, como os programadores são seus autores, precisam se preocupar na maneira com que ela será contada.
 - a. Para estruturar um código limpo, é necessário criar funções simples, claras e pequenas.



Clean code

Regras principais:

4. DRY (Don't Repeat Yourself): esta se aplica a diversas áreas de desenvolvimento, tais como Banco de Dados, Testes, Documentação, etc
 - a. O DRY diz que cada pedaço do conhecimento de um sistema deve ter uma representação única e ser totalmente livre de ambiguidades. Em outras palavras, define que não pode existir duas partes do programa que desempenhem a mesma função.



Clean code

Regras principais:

5. Comente o necessário, mas não exagere. Esse princípio afirma que comentários devem ser feitos, porém, se forem realmente necessários.
 - a. O que ocorre é que, enquanto os códigos são constantemente modificados, os comentários não. Eles são esquecidos e, portanto, deixam de retratar a funcionalidade real dos códigos.



Clean code

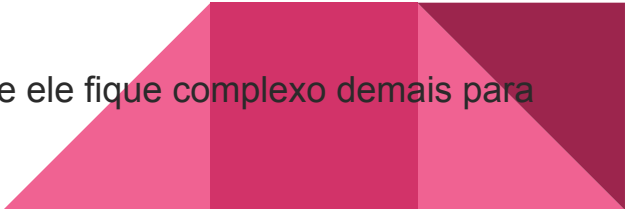
Regras principais:

6. Tratamento de erros. Seja responsável por garantir que o código fará o que precisa.
 - a. Faça testes de cada pedacinho separadamente e depois tudo junto.
 - b. Tratar as exceções de forma correta, mas não encher o programa de exceções.



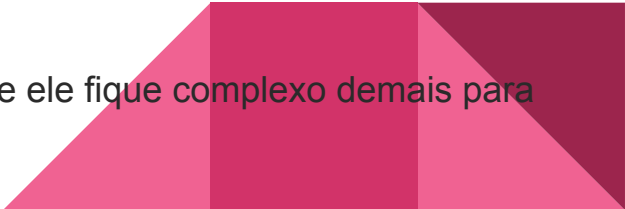
Como fazer testes?

Testar é uma etapa muito importante. Afinal, um código só é validado através de testes. Por isso, ele deve seguir algumas regras, como:

1. **Fast:** O teste deve ser rápido, permitindo que seja realizado várias vezes e a todo momento;
 2. **Independente:** a fim de evitar que cause efeito cascata quando da ocorrência de uma falha – o que dificulta a análise dos problemas;
 3. **Repetível:** Deve permitir a repetição do teste diversas vezes e em ambientes diferentes;
 4. **Self-Validation:** Os testes bem escritos retornam com as respostas true ou false, justamente para que a falha não seja subjetiva;
 5. O ideal é que sejam **escritos antes do próprio código**, pois evita que ele fique complexo demais para ser testado.
- 

Como fazer testes?

Testar é uma etapa muito importante. Afinal, um código só é validado através de testes. Por isso, ele deve seguir algumas regras, como:

1. **Fast:** O teste deve ser rápido, permitindo que seja realizado várias vezes e a todo momento;
 2. **Independente:** a fim de evitar que cause efeito cascata quando da ocorrência de uma falha – o que dificulta a análise dos problemas;
 3. **Repetível:** Deve permitir a repetição do teste diversas vezes e em ambientes diferentes;
 4. **Self-Validation:** Os testes bem escritos retornam com as respostas true ou false, justamente para que a falha não seja subjetiva;
 5. O ideal é que sejam **escritos antes do próprio código**, pois evita que ele fique complexo demais para ser testado.
- 

Exemplo de comentário

```
def handle_type_7_instruction (partes, comando, result):  
    """Processa uma instrução do tipo 7 e atualiza o resultado com a representação hexadecimal  
    correspondente.  
  
    Esta função realiza as seguintes etapas:  
    1. Inicializa uma representação binária de dois bits com `\"00\"`.  
    2. Atualiza o primeiro bit com `\"1\"` se o comando for `\"OUT\"`.  
    3. Atualiza o segundo bit com `\"1\"` se o segundo componente de `partes` for `\"ADDR\"`.  
    4. Converte o número do registrador (extraído de `partes[2]`) para uma representação binária de  
    dois dígitos.  
    5. Concatena a representação binária dos bits de controle e o registrador.  
    6. Converte a representação binária final para hexadecimal.  
    7. Adiciona o valor hexadecimal gerado à string `result`.  
  
    Parâmetros:  
  
    partes (list of str): Lista contendo os componentes da instrução. Por exemplo: [\"INSTRUCTION\",  
    \"ADDR\", \"R2\"].  
  
    comando (str): O comando da instrução, como `\"OUT\"`, que influencia a configuração dos bits de  
    controle.  
  
    result (str): A string que será atualizada com o valor hexadecimal gerado pela função.  
  
    Retorna:  
  
    str: A string `result` atualizada com o valor hexadecimal correspondente à instrução processada.  
  
    Exemplo:  
  
    >>> handle_type_7_instruction([\"INSTRUCTION\", \"ADDR\", \"R2\"], \"OUT\", \"A\")  
    'AE'  
  
    - Comando: \"OUT\" define o primeiro bit como 1.  
    - `partes[1]` é \"ADDR\", definindo o segundo bit como 1.  
    - O registrador \"R2\" é convertido para binário e concatenado.  
    - A representação binária final é \"1110\", que é convertida para o hexadecimal \"E\".  
    - O resultado final é \"AE\", retornado como a string atualizada.\"""
```

Exemplo de comentário

O que significa instrução to tipo 7?

```
def handle_type_7_instruction (partes, comando, result):  
    """Processa uma instrução do tipo 7 e atualiza o resultado com a representação hexadecimal  
    correspondente.  
  
    Esta função realiza as seguintes etapas:  
    1. Inicializa uma representação binária de dois bits com ``00``.  
    2. Atualiza o primeiro bit com ``1`` se o comando for ``OUT``.  
    3. Atualiza o segundo bit com ``1`` se o segundo componente de `partes` for ``ADDR``.  
    4. Converte o número do registrador (extraído de `partes[2]`) para uma representação binária de  
    dois dígitos.  
    5. Concatena a representação binária dos bits de controle e o registrador.  
    6. Converte a representação binária final para hexadecimal.  
    7. Adiciona o valor hexadecimal gerado à string `result`.  
  
    Parâmetros:  
  
    partes (list of str): Lista contendo os componentes da instrução. Por exemplo: ["INSTRUCTION",  
    "ADDR", "R2"].  
  
    comando (str): O comando da instrução, como ``OUT``, que influencia a configuração dos bits de  
    controle.  
  
    result (str): A string que será atualizada com o valor hexadecimal gerado pela função.  
  
    Retorna:  
  
    str: A string `result` atualizada com o valor hexadecimal correspondente à instrução processada.  
  
    Exemplo:  
  
    >>> handle_type_7_instruction(["INSTRUCTION", "ADDR", "R2"], "OUT", "A")  
    'AE'  
  
    - Comando: "OUT" define o primeiro bit como 1.  
    - `partes[1]` é "ADDR", definindo o segundo bit como 1.  
    - O registrador "R2" é convertido para binário e concatenado.  
    - A representação binária final é "1110", que é convertida para o hexadecimal "E".  
    - O resultado final é "AE", retornado como a string atualizada."""
```

Exemplo de nesting code

```
func main() {  
    if !areValidInstructions {  
        if areArithmetic {  
            if haveRegisters {  
                for _, instruction := range instruction_list {  
                    switch instruction.lower() {  
                        case "add":  
                            memory_list += '0x8'  
                        case "or":  
                            memory_list += '0xd'  
                        default:  
                            memory_list += '0x6'  
                    }  
                }  
            } else {  
                log.Fatalf("not regiter arguments")  
            }  
        } else {  
            log.Fatalf("not arithmetic instruction")  
        }  
    } else {  
        log.Fatalf("unrecognized instruction")  
    }  
}
```

Qual é o problema?



Exemplo de nesting code

```
func main() {  
    if !areValidInstructions {  
        if areArithmetic {  
            if haveRegisters {  
                for _, instruction := range instruction_list {  
                    switch instruction.lower() {  
                        case "add":  
                            memory_list += '0x8'  
                        case "or":  
                            memory_list += '0xd'  
                        default:  
                            memory_list += '0x6'  
                    }  
                }  
            } else {  
                log.Fatalf("not regiter arguments")  
            }  
        } else {  
            log.Fatalf("not arithmetic instruction")  
        }  
    } else {  
        log.Fatalf("unrecognized instruction")  
    }  
}
```

lógica principal

Qual é o problema?

Ender a lógica principal, ao mesmo tempo que mantemos na nossa memória todas as condições que devem ser cumpridas...

Exemplo de inversion

Ao invés de verificar se a condição é verdadeira (ou falsa), invertemos, e reduzimos alguns níveis

```
func main() {  
    if !areValidInstruction {  
        if areArithmetic {  
            if hasRegisters {  
                for _, instruction := range instruction_list {  
                    switch instruction.lower() {  
                        case "add":  
                            memory_list += '0x8'  
                        case "or":  
                            memory_list += '0xd'  
                        default:  
                            memory_list += '0x6'  
                    }  
                }  
            } else {  
                log.Fatalf("not register arguments")  
            }  
        } else {  
            log.Fatalf("not arithmetic instruction")  
        }  
    } else {  
        log.Fatalf("unrecognized instruction")  
    }  
}
```



```
func main() {  
    if areValidInstruction { log.Fatalf("unrecognized instruction") }  
    if areArithmetic {  
        if haveRegisters {  
            for _, instruction := range instruction_list {  
                switch instruction.lower() {  
                    case "add":  
                        memory_list += '0x8'  
                    case "or":  
                        memory_list += '0xd'  
                    default:  
                        memory_list += '0x6'  
                }  
            }  
        } else {  
            log.Fatalf("not register arguments")  
        }  
    } else {  
        log.Fatalf("not arithmetic instruction")  
    }  
}
```

Exemplo de inversion

Podemos repetir a lógica para todos os IFs (assumimos que `log.Fatalf` Causa um return)

```
func main() {  
    if areValidInstruction { log.Fatalf("unrecognized instruction" ) }  
  
    if !areArithmetic { log.Fatalf("not arithmetic instruction" ) }  
  
    if !haveRegisters { log.Fatalf("not register arguments" ) }  
  
    for _, instruction := range instruction_list {  
        switch instruction.lower() {  
            case "add":  
                memory_list += `0x8`  
            case "or":  
                memory_list += `0xd`  
            default:  
                memory_list += `0x6`  
        }  
    }  
}
```

Ajuda a pensar que, se o código alcança este ponto, então as linhas anteriores não precisam ser uma preocupação



Exemplo de Relacionar IF conditions

Se as três condições têm que ser verdadeiras, então poderia ser uma única condição:

```
func main() {  
    if areValidInstruction { log.Fatalf("unrecognized instruction" ) }  
  
    if !areArithmetic { log.Fatalf("not arithmetic instruction" ) }  
  
    if !haveRegisters { log.Fatalf("not regiter arguments" ) }  
  
    for _, instruction := range instruction_list {  
        switch instruction.lower() {  
            case "add":  
                memory_list += `0x8`  
            case "or":  
                memory_list += `0xd`  
            default:  
                memory_list += `0x6`  
        }  
    }  
}
```

Exemplo de Relacionar IF conditions

Se as três condições têm que ser verdadeiras, então poderia ser uma única condição:

```
func main() {  
    if areValidInstruction and !areArithmetic and !haveRegisters {  
        log.Fatalf("unrecognized instruction") }  
  
    for _, instruction := range instruction_list {  
        switch instruction.lower() {  
            case "add":  
                memory_list += '0x8'  
            case "or":  
                memory_list += '0xd'  
            default:  
                memory_list += '0x6'  
        }  
    }  
}
```


Cuidado: se perde granularidade
nos comentários relacionados às
falhas.

Exemplo de Extraction

Empacotar em sub-funções

```
func main() {  
    if check_valid_instruction() {  
        log.Fatalf("unrecognized instruction") }  
  
    for _, instruction := range instruction_list {  
        switch instruction.lower() {  
            case "add":  
                memory_list += '0x8'  
            case "or":  
                memory_list += '0xd'  
            default:  
                memory_list += '0x6'  
        }  
    }  
}
```

```
bool func check_valid_list_of_instructions() {  
    if areValidInstruction and !areArithmetic  
    and !haveRegisters {  
        return 1 }  
    else { return 0 }  
}
```



Exemplo de Extraction

Empacotar em sub-funções

```
func main() {  
    if check_valid_list_of_instructions() {  
        log.Fatalf("unrecognized instruction") }  
  
    for _, instruction := range instruction_list {  
        Add_instruction_to_memory(instruction)  
    }  
}
```

```
func Add_instruction_to_memory(instruction) {  
    switch instruction.lower() {  
        case "add":  
            memory_list += '0x8'  
        case "or":  
            memory_list += '0xd'  
        default:  
            memory_list += '0x6'  
    }  
}
```

Resultado

Agora este código é facilmente legível. Só olhando podemos perceber que este código verifica se a instrução é válida e agrega esta à lista de instruções na memória.

```
func main() {  
    if check_valid_list_of_instructions() {  
        log.Fatalf("unrecognized instruction") }  
  
    for _, instruction := range instruction_list {  
        Add_instruction_to_memory(instruction)  
    }  
}
```



Evitar repetições, qual é o problema?

```
func getUser(w http.ResponseWriter, r *http.Request) {
    userID := r.URL.Path[len("/user/"):]:]

    cacheMux.Lock()
    user, found := cache[userID]
    cacheMux.Unlock()

    if !found {
        query := "SELECT user_name, email FROM users WHERE user_id = ?"
        stmt, _ := db.Prepare(query)

        var u User
        _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)

        cacheMux.Lock()
        cache[userID] = u
        cacheMux.Unlock()

        user = u
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(user)
}
```

```
func getUsers(w http.ResponseWriter, r *http.Request) {
    var requestBody RequestBody
    json.NewDecoder(r.Body).Decode(&requestBody)
    userIDs := requestBody.UserIDs
    var users []User

    for _, userID := range userIDs {
        cacheMux.Lock()
        user, found := cache[userID]
        cacheMux.Unlock()

        if !found {
            query := "SELECT user_name, email FROM users WHERE user_id = ?"
            stmt, _ := db.Prepare(query)

            var u User
            _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)

            cacheMux.Lock()
            cache[userID] = u
            cacheMux.Unlock()

            user = u
        }

        users = append(users, user)
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(users)
}
```

https://www.youtube.com/watch?v=-AzSRHiV9Cc&ab_channel=KantanCoding

A mesma lógica em duas funções diferentes

```
func getUser(w http.ResponseWriter, r *http.Request) {
```

```
    userID := r.URL.Path[len("/user/"):]
```

```
    cacheMux.Lock()
```

```
    user, found := cache[userID]
```

```
    cacheMux.Unlock()
```

```
    if !found {
```

```
        query := "SELECT user_name, email FROM users WHERE user_id = ?"
```

```
        stmt, _ := db.Prepare(query)
```

```
        var u User
```

```
        _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)
```

```
        cacheMux.Lock()
```

```
        cache[userID] = u
```

```
        cacheMux.Unlock()
```

```
        user = u
```

```
    }
```

```
    w.Header().Set("Content-Type", "application/json")
```

```
    json.NewEncoder(w).Encode(user)
```

```
func getUsers(w http.ResponseWriter, r *http.Request) {
```

```
    var requestBody RequestBody
```

```
    json.NewDecoder(r.Body).Decode(&requestBody)
```

```
    userIDs := requestBody.UserIDs
```

```
    var users []User
```

```
    for _, userID := range userIDs {
```

```
        cacheMux.Lock()
```

```
        user, found := cache[userID]
```

```
        cacheMux.Unlock()
```

```
        if !found {
```

```
            query := "SELECT user_name, email FROM users WHERE user_id = ?"
```

```
            stmt, _ := db.Prepare(query)
```

```
            var u User
```

```
            _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)
```

```
            cacheMux.Lock()
```

```
            cache[userID] = u
```

```
            cacheMux.Unlock()
```

```
            user = u
```

```
        }
```

```
        users = append(users, user)
```

```
    }
```

```
    w.Header().Set("Content-Type", "application/json")
```

```
    json.NewEncoder(w).Encode(users)
```

```
}
```

https://www.youtube.com/watch?v=-AzSRHiV9Cc&ab_channel=KantanCoding

A mesma lógica em duas funções diferentes

```
func getUser(w http.ResponseWriter, r *http.Request) {
    userID := r.URL.Path[len("/user/"): ]

    cacheMux.Lock()
    user, found := cache[userID]
    cacheMux.Unlock()

    if !found {
        query := "SELECT user_name, email FROM users WHERE user_id = ?"
        stmt, _ := db.Prepare(query)

        var u User
        _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)

        cacheMux.Lock()
        cache[userID] = u
        cacheMux.Unlock()

        user = u
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(user)
}
```

```
func getUsers(w http.ResponseWriter, r *http.Request) {
    var requestBody RequestBody
    json.NewDecoder(r.Body).Decode(&requestBody)
    userIDs := requestBody.UserIDs
    var users []User

    for _, userID := range userIDs {
        cacheMux.Lock()
        user, found := cache[userID]
        cacheMux.Unlock()

        if !found {
            query := "SELECT user_name, email FROM users WHERE user_id = ?"
            stmt, _ := db.Prepare(query)

            var u User
            _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)

            cacheMux.Lock()
            cache[userID] = u
            cacheMux.Unlock()

            user = u
        }

        users = append(users, user)
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(users)
}
```

https://www.youtube.com/watch?v=-AzSRHiv9Cc&ab_channel=KantanCoding

Usar funções para evitar repetir código

```
func getSingleUser(userID string) User {  
    cacheMux.Lock()  
    user, found := cache[userID]  
    cacheMux.Unlock()  
  
    if !found {  
        query := "SELECT user_name, email FROM users WHERE user_id = ?"  
        stmt, _ := db.Prepare(query)  
  
        var u User  
        _ = stmt.QueryRow(userID).Scan(&u.Username, &u.Email)  
  
        cacheMux.Lock()  
        cache[userID] = u  
        cacheMux.Unlock()  
  
        user = u  
    }  
    return user  
}
```

```
func writeResponse(w http.ResponseWriter, response interface{}) {  
    w.Header().Set("Content-Type", "application/json")  
    json.NewEncoder(w).Encode(response)  
}
```

```
func getUsers(w http.ResponseWriter, r *http.Request) {  
    var requestBody RequestBody  
    json.NewDecoder(r.Body).Decode(&requestBody)  
    userIDs := requestBody.UserIDs  
    var users []User  
  
    for _, userID := range userIDs {  
        user := getSingleUser(userID)  
        users = append(users, user)  
    }  
  
    writeResponse(w, users)  
}  
  
func getUser(w http.ResponseWriter, r *http.Request) {  
    userID := r.URL.Path[len("/user/"):]  
    user := getSingleUser(userID)  
  
    writeResponse(w, user)  
}
```

https://www.youtube.com/watch?v=-AzSRHiv9Cc&ab_channel=KantanCoding

Atividade

Reescrever o montador usando as técnicas explicadas

